

LITERATURE

To order Intel Literature or obtain literature pricing information in the U.S. and Canada call or write Intel Literature Sales. In Europe and other international locations, please contact your **local** sales office or distributor.

INTEL LITERATURE SALES
P.O. BOX 7641
Mt. Prospect, IL 60056-7641

In the U.S. and Canada
call toll free
(800) 548-4725

CURRENT HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information. All handbooks can be ordered individually, and most are available in a pre-packaged set in the U.S. and Canada.

TITLE	INTEL ORDER NUMBER	ISBN
SET OF THIRTEEN HANDBOOKS (Available in U.S. and Canada)	231003	N/A
CONTENTS LISTED BELOW FOR INDIVIDUAL ORDERING:		
COMPONENTS QUALITY/RELIABILITY	210997	1-55512-132-2
EMBEDDED APPLICATIONS	270648	1-55512-123-3
8-BIT EMBEDDED CONTROLLERS	270645	1-55512-121-7
16-BIT EMBEDDED CONTROLLERS	270646	1-55512-120-9
16/32-BIT EMBEDDED PROCESSORS	270647	1-55512-122-5
MEMORY PRODUCTS	210830	1-55512-117-9
MICROCOMMUNICATIONS	231658	1-55512-119-5
MICROCOMPUTER PRODUCTS	280407	1-55512-118-7
MICROPROCESSORS	230843	1-55512-115-2
PACKAGING	240800	1-55512-128-4
PERIPHERAL COMPONENTS	296467	1-55512-127-6
PRODUCT GUIDE (Overview of Intel's complete product lines)	210846	1-55512-116-0
PROGRAMMABLE LOGIC	296083	1-55512-124-1
ADDITIONAL LITERATURE: (Not included in handbook set)		
AUTOMOTIVE HANDBOOK	231792	1-55512-125-x
INTERNATIONAL LITERATURE GUIDE (Available in Europe only)	E00029	N/A
CUSTOMER LITERATURE GUIDE	210620	N/A
MILITARY HANDBOOK (2 volume set)	210461	1-55512-126-8
SYSTEMS QUALITY/RELIABILITY	231762	1-55512-046-6

Embedded Applications

How Intel's 8-, 16- and 32-bit embedded controllers and processors are put to work is documented in this collection of application notes. This edition contains over 50 application notes and article reprints on topics including the new i960 32-bit RISC-based embedded processors, MCS[®]-48, MCS-51 and MCS-96 families. The 80186/80188 family and a general section on microcontrollers are also covered.

Charts, diagrams, technical specifications and architectural information are helpful tools for the designer, but of even more use are application techniques. Experience shared is the best teacher. From laser printers to production control, the notes contained in this handbook discuss hardware and software implementation and present helpful design techniques.



Intel Corporation is a leading supplier of microcomputer components, modules and systems. When Intel invented the microprocessor in 1971, it created the era of the microcomputer. Today, Intel architectures are considered world standards. Whether used in embedded applications such as automobiles, printers and microwave ovens, or as the CPU in personal computers, client servers or supercomputers, Intel delivers leading-edge technology.

EMBEDDED CONTROLLER APPLICATIONS HANDBOOK

1991

About Our Cover:
Thinkers, inventors, and artists throughout history have breathed life into their ideas by converting them into rough working sketches, models, and products. This series of covers shows a few of these creations, along with the applications and products created by Intel customers.



Intel Corporation is a leading supplier of microcomputer components, modules and systems. When Intel invented the microprocessor in 1971, it created the era of the microcomputer. Today, Intel architectures are considered world standards. Whether used in embedded applications such as automobiles, printers and microwave ovens, or as the CPU in personal computers, client servers or supercomputers, Intel delivers leading-edge technology.

EMBEDDED CONTROLLER APPLICATIONS

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel products:

287, 376, 386, 387, 486, 4-SITE, Above, ACE51, ACE96, ACE186, ACE196, ACE960, ActionMedia, BITBUS, COMMputer, CREDIT, Data Pipeline, DVI, ETOX, FaxBACK, Genius, i, i⁺, i486, i750, i860, ICE, iCEL, ICEVIEW, iCS, iDBP, iDIS, i²ICE, iLBX, iMDDX, iMMX, Inboard, Insite, Intel, intel, Intel386, intelBOS, Intel Certified, InteleVision, intelligent Identifier, intelligent Programming, Inteltec, Intellink, iOSP, iPAT, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, Library Manager, MAPNET, MCS, Megachassis, MICROMAINFRAME, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, Pro750, PROMPT, Promware, QUEST, QueX, Quick-Erase, Quick-Pulse Programming, READY LAN, RMX/80, RUPI, Seamless, SLD, SugarCube, SX, ToolTALK, UPI, VAPI, Visual Edge, VLSiCEL, and ZapCode, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

CHMOS and HMOS are patented processes of Intel Corp.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

CUSTOMER SUPPORT

INTEL'S COMPLETE SUPPORT SOLUTION WORLDWIDE

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, consulting services and network management services. For detailed information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is extensive. It can start with assistance during your development effort to network management. 100 Intel sales and service offices are located worldwide—in the U.S., Canada, Europe and the Far East. So wherever you're using Intel technology, our professional staff is within close reach.

HARDWARE SUPPORT SERVICES

Intel's hardware maintenance service, starting with complete on-site installation will boost your productivity from the start and keep you running at maximum efficiency. Support for system or board level products can be tailored to match your needs, from complete on-site repair and maintenance support to economical carry-in or mail-in factory service.

Intel can provide support service for not only Intel systems and emulators, but also support for equipment in your development lab or provide service on your product to your end-user/customer.

SOFTWARE SUPPORT SERVICES

Software products are supported by our Technical Information Service (TIPS) that has a special toll free number to provide you with direct, ready information on known, documented problems and deficiencies, as well as work-arounds, patches and other solutions.

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and; *COMMENTS Magazine*). Basic support consists of updates and the subscription service. Contracts are sold in environments which represent product groupings (e.g., iRMX® environment).

NETWORK SERVICE AND SUPPORT

Today's broad spectrum of powerful networking capabilities are only as good as the customer support provided by the vendor. Intel offers network services and support structured to meet a wide variety of end-user computing needs. From a ground up design of your network's physical and logical design to implementation, installation and network wide maintenance. From software products to turn-key system solutions; Intel offers the customer a complete networked solution. With over 10 years of network experience in both the commercial and Government arena; network products, services and support from Intel provide you the most optimized network offering in the industry.

CONSULTING SERVICES

Intel provides field system engineering consulting services for any phase of your development or application effort. You can use our system engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training and customizing an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, BITBUS™ and LAN applications.

DATA SHEET DESIGNATIONS

Intel uses various data sheet markings to designate each phase of the document as it relates to the product. The marking appears in the upper, right-hand corner of the data sheet. The following is the definition of these markings:

Data Sheet Marking	Description
Product Preview	Contains information on products in the design phase of development. Do not finalize a design with this information. Revised information will be published when the product becomes available.
Advanced Information	Contains information on products being sampled or in the initial production phase of development.*
Preliminary	Contains preliminary information on new products in production.*
No Marking	Contains information on products in full production.*

*Specifications within these data sheets are subject to change without notice. Verify with your local Intel sales office that you have the latest data sheet before finalizing a design.

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and COMEYVZ Manuals). Basic support consists of updates and the subscription service. Contracts are sold in environments which represent product lineups (e.g., IBM® environment).

NETWORK SERVICE AND SUPPORT

Today's broad spectrum of powerful networking capabilities are only as good as the customer support provided by the vendor. Intel offers network services and support structured to meet a wide variety of end-user computing needs. From a ground up design of your network's physical and logical design to implementation, installation and network wide maintenance. From software products to turn-key system solutions, Intel offers the customer a complete networked solution. With over 10 years of network experience in both the commercial and Government sectors, network products, services and support from Intel provide you the most optimized network offering in the industry.

CONSULTING SERVICES

Intel provides field system engineering consulting services for any phase of your development or application effort. You can use our system engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training and customizing an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcomputers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. Just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, BIOS™, and LAN applications.



MCS®-48 Application Notes

1

**MCS®-51 Application Notes &
Article Reprints**

2

**ASIC Family Application Note
& Article Reprint**

3

RUPITM Application Notes

4

80186/188 Application Notes

5

**MCS®-96 Application Notes &
Article Reprint**

6

MCS®-96 Diagnostic Library

7

80960 Article Reprints

8

**General Microcontroller
Application Notes**

9

1	MCS®-48 Application Notes
2	MCS®-51 Application Notes & Article Reprints
3	ASIC Family Application Note & Article Reprint
4	RUBITM Application Notes
5	80186/188 Application Notes
6	MCS®-96 Application Notes & Article Reprint
7	MCS®-96 Diagnostic Library
8	80960 Article Reprints
9	General Microcontroller Application Notes

Table of Contents

Alphanumeric Index	xi
MCS-48 FAMILY	
Chapter 1	
MCS®-48 APPLICATION NOTES	
AP-24 Application Techniques for the MCS®-48 Family	1-1
AP-40 Keyboard/Display Scanning with Intel's MCS®-48 Microcomputers	1-25
AP-49 Serial I/O and Math Utilities for the 8049 Microcomputers	1-50
AP-55A A High-Speed Emulator for the Intel MCS®-48 Microcomputers	1-73
AP-91 Using the 8049 as an 80 Column Printer Controller	1-173
MCS-51 FAMILY	
Chapter 2	
MCS®-51 APPLICATION NOTES & ARTICLE REPRINTS	
AP-69 An introduction to the Intel MCS®-51 Single-Chip Microcomputer	2-1
AP-70 Using the Intel MCS-51 Boolean Processing Capabilities	2-31
AP-223 8051 Based CRT Terminal Controller	2-77
AB-38 Interfacing the 82786 Graphics Coprocessor to the 8051	2-154
AB-39 Interfacing the Densitron LCD to the 8051	2-161
AB-40 32-Bit Math Routines for the 8051	2-169
AB-12 Designing a Mailbox Memory for Two 80C31 Microcontrollers Using EPLDs	2-179
AP-252 Designing With The 80C51BH	2-188
AP-410 Enhanced Serial Port on the 83C51FA	2-213
AB-41 Software Serial Port Implemented with the PCA	2-221
AP-415 83C51FA/FB PCA Cookbook	2-245
AP-425 Small DC Motor Control	2-290
AR-517 Using the 8051 with Resonant Transducers	2-305
AR-526 Analog/Digital Processing with Microcontrollers	2-310
Chapter 3	
ASIC FAMILY APPLICATION NOTE & ARTICLE REPRINT	
AP-413 Using Intel's ASIC Core Cell to Expand the Capabilities of an 80C51-Based System	3-1
AR-537 A Fast-Turnaround, Easily Testable ASIC Chip for Serial Bus Control	3-12
THE RUPI FAMILY	
Chapter 4	
RUPI™ APPLICATION NOTES	
AP-281 UPI-452 Accelerates iAPX 286 Bus Performance	4-1
AP-283 Flexibility in Frame Size with the 8044	4-23
80186/80188 FAMILY	
Chapter 5	
80186/188 APPLICATION NOTES	
AP-258 High Speed Numerics with the 80186/80188 and 8087	5-1
AP-286 80186/188 Interface to Intel Microcontrollers	5-19
AB-36 80186/80188 DMA Latency	5-51
AB-37 80186/80188 EFI Drive and Oscillator Operation	5-55
AB-31 The 80C186/80C188 Integrated Refresh Control Unit	5-58
AB-35 DRAM Refresh/Control with the 80186/80188	5-72
MCS-96 FAMILY	
Chapter 6	
MCS®-96 APPLICATION NOTES & ARTICLE REPRINT	
AP-248 Using The 8096	6-1

Table of Contents (Continued)

AP-275 An FFT Algorithm for MCS-96 Products Including Supporting Routines and Examples	6-106
AB-32 Upgrade Path from 8096-90 to 8096BH to 80C196	6-183
AB-33 Memory Expansion for the 8096	6-187
AB-34 Integer Square Root Routine for the 8096	6-200
AP-406 MCS-96 Analog Acquisition Primer	6-205
AR-515 A Single-Chip Image Processor	6-306
Chapter 7	
MCS®-96 Diagnostic Library	
MCS®-96 Diagnostic Library	7-1
Chapter 8	
80960 ARTICLE REPRINTS	
AR-541 Intel's 80960: An Architecture Optimized for Embedded Control	8-1
GENERAL MICROCONTROLLER	
Chapter 9	
APPLICATION NOTES	
AP-125 Designing Microcontroller Systems for Electrically Noisy Environments	9-1
AP-155 Oscillators for Microcontrollers	9-24

Alphanumeric Index

AB-12 Designing a Mailbox Memory for Two 80C31 Microcontrollers Using EPLDs	2-179
AB-31 The 80C186/80C188 Integrated Refresh Control Unit	5-58
AB-32 Upgrade Path from 8096-90 to 8096BH to 80C196	6-183
AB-33 Memory Expansion for the 8096	6-187
AB-34 Integer Square Root Routine for the 8096	6-200
AB-35 DRAM Refresh/Control with the 80186/80188	5-72
AB-36 80186/80188 DMA Latency	5-51
AB-37 80186/80188 EFI Drive and Oscillator Operation	5-55
AB-38 Interfacing the 82786 Graphics Coprocessor to the 8051	2-154
AB-39 Interfacing the Densitron LCD to the 8051	2-161
AB-40 32-Bit Math Routines for the 8051	2-169
AB-41 Software Serial Port Implemented with the PCA	2-221
AP-125 Designing Microcontroller Systems for Electrically Noisy Environments	9-1
AP-155 Oscillators for Microcontrollers	9-24
AP-223 8051 Based CRT Terminal Controller	2-77
AP-24 Application Techniques for the MCS®-48 Family	1-1
AP-248 Using The 8096	6-1
AP-252 Designing With The 80C51BH	2-188
AP-258 High Speed Numerics with the 80186/80188 and 8087	5-1
AP-275 An FFT Algorithm for MCS-96 Products Including Supporting Routines and Examples	6-106
AP-281 UPI-452 Accelerates iAPX 286 Bus Performance	4-1
AP-283 Flexibility in Frame Size with the 8044	4-23
AP-286 80186/188 Interface to Intel Microcontrollers	5-19
AP-40 Keyboard/Display Scanning with Intel's MCS®-48 Microcomputers	1-25
AP-406 MCS-96 Analog Acquisition Primer	6-205
AP-410 Enhanced Serial Port on the 83C51FA	2-213
AP-413 Using Intel's ASIC Core Cell to Expand the Capabilities of an 80C51-Based System	3-1
AP-415 83C51FA/FB PCA Cookbook	2-245
AP-425 Small DC Motor Control	2-290
AP-49 Serial I/O and Math Utilities for the 8049 Microcomputers	1-50
AP-55A A High-Speed Emulator for the Intel MCS®-48 Microcomputers	1-73
AP-69 An introduction to the Intel MCS®-51 Single-Chip Microcomputer	2-1
AP-70 Using the Intel MCS-51 Boolean Processing Capabilities	2-31
AP-91 Using the 8049 as an 80 Column Printer Controller	1-173
AR-515 A Single-Chip Image Processor	6-306
AR-517 Using the 8051 with Resonant Transducers	2-305
AR-526 Analog/Digital Processing with Microcontrollers	2-310
AR-537 A Fast-Turnaround, Easily Testable ASIC Chip for Serial Bus Control	3-12
AR-541 Intel's 80960: An Architecture Optimized for Embedded Control	3-1
MCS®-96 Diagnostic Library	7-1

MCS®-48 Application Notes

1

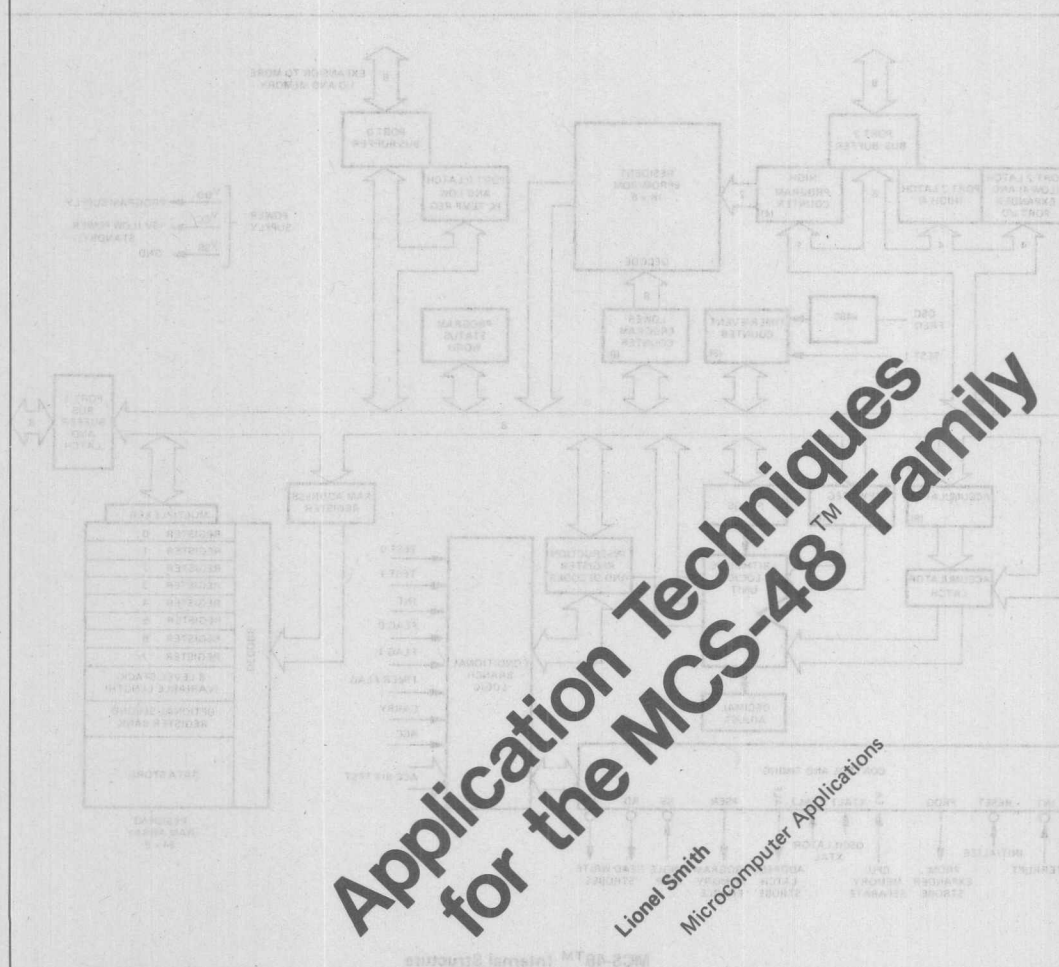
1

The processors in the MC248 family all share an identical architecture. The only significant difference is the type of on-board program storage which is provided. The 8148 (see Figure 1) includes 8032 bytes of erasable, programmable ROM (EPROM). The 8032 replaces the EPROM with an erasable amount of mask programmed ROM, and the 8032 provides the CPU function with no on-board program storage. All three of these processors

February 1977

February 1977

1



Lionel Smith
Mic

Smith
Microcomputer Applications

INTRODUCTION

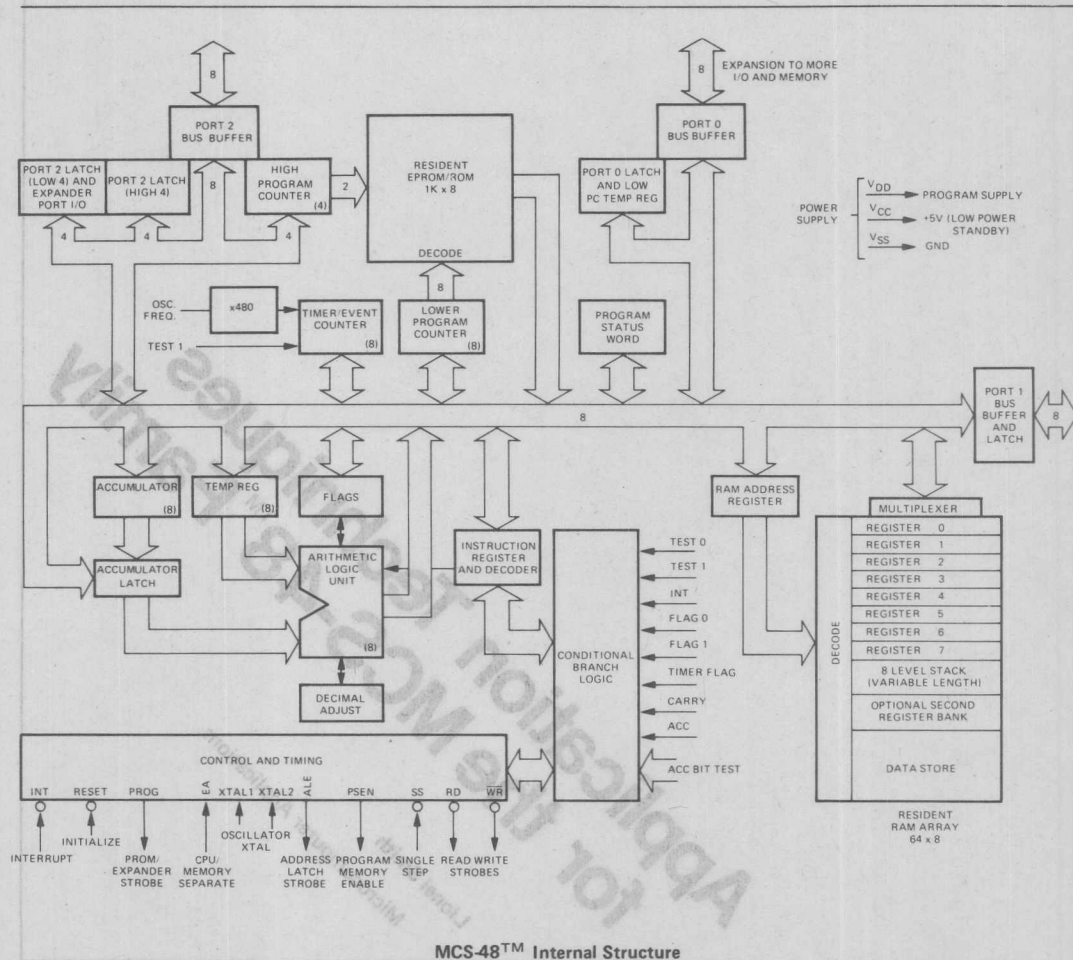
The INTEL[®] MCS-48[™] family consists of a series of seven parts, including three processors, which take advantage of the latest advances in silicon technology to provide the system designer with an effective solution to a wide variety of design problems. The significant contribution of the MCS-48 family is that instead of consisting of integrated microcomputer components it consists of integrated microcomputer systems. A single integrated circuit contains the processor, RAM, ROM (or PROM), a timer, and I/O.

This application note suggests a variety of application techniques which are useful with the MCS-48. Rather than presenting the design of a complete system it describes the implementation of "sub-systems" which are common to many micropro-

cessor based systems. The subsystems described are analog input and output, the use of tables for function evaluation, receiving serial code, transmitting serial code, and parity generation. After an overview of the MCS-48 family these areas are discussed in a more or less independent manner.

THE MCS-48™ FAMILY

The processors in the MCS-48 family all share an identical architecture. The only significant difference is the type of on board program storage which is provided. The 8748 (see Figure 1) includes 1024 bytes of erasable, programmable, ROM (EPROM), the 8048 replaces the EPROM with an equivalent amount of mask programmed ROM, and the 8035 provides the CPU function with no on board program storage. All three of these processors



	Mnemonic	Description	Bytes	Cycle		Mnemonic	Description	Bytes	Cycles	
Accumulator	ADD A,R	Add register to A	1	1	Subroutine	CALL	Jump to subroutine	2	2	
	ADD A,@R	Add data memory to A	1	1		RET	Return	1	2	
	ADD A,#data	Add immediate to A	2	2		RETR	Return and restore status	1	2	
	ADDC A,R	Add register with carry	1	1	Flags	CLR C	Clear Carry	1	1	
	ADDC A,@R	Add data memory with carry	1	1		CPL C	Complement Carry	1	1	
	ADDC A,#data	Add immediate with carry	2	2		CLR F0	Clear Flag 0	1	1	
	ANL A,R	And register to A	1	1		CPL F0	Complement Flag 0	1	1	
	ANL A,@R	And data memory to A	1	1		CLR F1	Clear Flag 1	1	1	
	ANL A,#data	And immediate to A	2	2		CPL F1	Complement Flag 1	1	1	
	ORL A,R	Or register to A	1	1	Data Movers	MOV A,R	Move register to A	1	1	
	ORL A,@R	Or data memory to A	1	1		MOV A,@R	Move data memory to A	1	1	
	ORL A,#data	Or immediate to A	2	2		MOV A,#data	Move immediate to A	2	2	
	XRL A,R	Exclusive Or register to A	1	1		MOV R,A	Move A to register	1	1	
	XRL A,@R	Exclusive or data memory to A	1	1		MOV @R,A	Move A to data memory	1	1	
	XRL A,#data	Exclusive or immediate to A	2	2		MOV R,#data	Move immediate to register	2	2	
	INC A	Increment A	1	1		MOV @R,#data	Move immediate to data memory	2	2	
	DEC A	Decrement A	1	1		MOV A,PSW	Move PSW to A	1	1	
	CLR A	Clear A	1	1		MOV PSW,A	Move A to PSW	1	1	
	CPL A	Complement A	1	1		XCH A,R	Exchange A and register	1	1	
	DA A	Decimal Adjust A	1	1		XCH A,@R	Exchange A and data memory	1	1	
	SWAP A	Swap nibbles of A	1	1		XCHD A,@R	Exchange nibble of A and register	1	1	
	RL A	Rotate A left	1	1		MOVX A,@R	Move external data memory to A	1	2	
	RLC A	Rotate A left through carry	1	1		MOVX @R,A	Move A to external data memory	1	2	
	RR A	Rotate A right	1	1		MOVP A,@A	Move to A from current page	1	2	
	RRC A	Rotate A right through carry	1	1		MOVP3 A,@A	Move to A from Page 3	1	2	
Input/Output	IN A,P	Input port to A	1	2	Timer/Counter	MOV A,T	Read Timer/Counter	1	1	
	OUTL P,A	Output A to port	1	2		MOV T,A	Load Timer/Counter	1	1	
	ANL P,#data	And immediate to port	2	2		STRT T	Start Timer	1	1	
	ORL P,#data	Or immediate to port	2	2		STRT CNT	Start Counter	1	1	
	INS A,BUS	Input BUS to A	1	2		STOP TCNT	Stop Timer/Counter	1	1	
	OUTL BUS,A	Output A to BUS	1	2		EN TCNTI	Enable Timer/Counter Interrupt	1	1	
	ANL BUS,#data	And immediate to BUS	2	2		DIS TCNTI	Disable Timer/Counter Interrupt	1	1	
	ORL BUS,#data	Or immediate to BUS	2	2	Control	EN I	Enable external interrupt	1	1	
	MOVD A,P	Input Expander port to A	1	2		DIS I	Disable external interrupt	1	1	
	MOVD P,A	Output A to Expander port	1	2		SEL RB0	Select register bank 0	1	1	
ANLD P,A	And A to Expander port	1	2	SEL RB1		Select register bank 1	1	1		
ORLD P,A	Or A to Expander port	1	2	SEL MB0		Select memory bank 0	1	1		
				SEL MB1		Select memory bank 1	1	1		
				ENTO CLK		Enable Clock output on T0	1	1		
				NOP		No Operation	1	1		
Registers	INC R	Increment register	1	1	Mnemonics copyright Intel Corporation 1976					
	INC @R	Increment data memory	1	1						
	DEC R	Decrement register	1	1						
	Branch	JMP addr	Jump unconditional	2	2					
		JMPP @A	Jump indirect	1	2					
		DJNZ R,addr	Decrement register and skip	2	2					
		JC addr	Jump on Carry = 1	2	2					
		JNC addr	Jump on Carry = 0	2	2					
		JZ addr	Jump on A Zero	2	2					
		JNZ addr	Jump on A not Zero	2	2					
		JTO addr	Jump on T0 = 1	2	2					
		JNT0 addr	Jump on T0 = 0	2	2					
		JT1 addr	Jump on T1 = 1	2	2					
		JNT1 addr	Jump on T1 = 0	2	2					
		JF0 addr	Jump on F0 = 1	2	2					
		JF1 addr	Jump on F1 = 1	2	2					
JTF addr		Jump on timer flag	2	2						
JNI addr		Jump on INT = 0	2	2						
JBb addr		Jump on Accumulator Bit	2	2						

Figure 2. 8048/8748/8035 Instruction Set

operate from a single 5-volt power supply. The 8748 requires an additional 25-volt supply only while the on board EPROM is being programmed. When installed in a system only the 5-volt supply is needed. Aside from program storage, these chips include 64 bytes of data storage (RAM), an eight bit timer which can also be used to count external events, 27 programmable I/O pins and the processor itself. The processor offers a wide range of instruction capability including many designed for bit, nibble, and byte manipulation. The instruction set is summarized in Figure 2.

Aside from the processors, the MCS-48 family includes 4 devices: one pure I/O device and 3 combination memory and I/O devices. The pure I/O device is the 8243, a device which is connected to a special 4 bit bus provided by the MCS-48 processors and which provides 16 I/O pins which can be programmatically controlled.

The combination memory and I/O devices consist of the 8355, the 8755, and the 8155. The 8355 and the 8755 both provide 2,048 bytes of program storage and two eight bit data ports. The only difference between these devices is that the 8355 contains masked program ROM and the 8755 contains EPROM. The 8155 combines 256 bytes of data storage (RAM), two eight bit data ports, a six bit control port, and a 14 bit programmable timer.

Figure 3 shows the various system configurations which can be achieved using the MCS-48 family of parts. It should also be noted that eight of the processors' I/O lines have been configured as a bidirectional bus which can be used to interface to standard Intel peripheral parts such as the 8251 USART (for serial I/O), the 8255A PPI (provides 24 I/O lines) and the complete range of memory components.

More detailed information concerning the MCS-48 family can be obtained from the "MCS-48 Microcomputer User's Manual" which provides a complete description of the MCS-48 family and its members. A general familiarity with this document will make the application techniques which follow easier to understand.

ANALOG I/O

If analog I/O is required for a MCS-48™ system there are many alternatives available from the makers of analog I/O modules. By searching through their catalogs it is possible to find almost any combination of features which is technically feasible. Perhaps the best example of such modules are the MP-10 and MP-20 hybrid modules recently introduced by Burr-Brown Research Corporation. The MP-10 provides two analog outputs and the MP-20 provides 16 analog inputs. Both of these units were

[] Number of Available Timers
() Number of Available I/O Lines

DATA MEMORY (RAM)	1K			
	8048	8035	8048	8035
1088	8048	8355	8355	2-8355
832	4-8155	4-8155	4-8155	4-8155
768	[5] (101)	[5] (116)	[5] (116)	[5] (131)
512	256			
	8048	8035	8048	8035
578	8048	8355	8355	2-8355
512	3-8155	3-8155	3-8155	3-8155
320	[4] (80)	[4] (95)	[4] (95)	[4] (110)
256	128			
	8048	8035	8048	8035
256	8048	8355	8355	2-8355
128	2-8155	2-8155	2-8155	2-8155
64	[3] (59)	[3] (74)	[3] (74)	[3] (89)
64	32			
	8048	8035	8048	8035
32	8048	8355	8355	2-8355
16	1-8155	1-8155	1-8155	1-8155
8	[2] (38)	[2] (53)	[2] (53)	[2] (68)
8	4			
	8048	8035	8048	8035
4	8048	8355	8355	2-8355
2	[1] (24)	[1] (28)	[1] (28)	[1] (43)
PROGRAM MEMORY (ROM)				
1K 2K 3K 4K				

Figure 3. The Expanded MCS-48™ System

specifically designed to interface with microprocessors.

A block diagram of the MP-10 is shown in Figure 4. It consists of two eight bit digital to analog converters, two eight bit latches which are loaded from the data bus, and address decoding logic to determine when the latches should be loaded. The D/A converters each generate an analog output in the range of 10 volts with an output impedance of 15Ω. Accuracy is ±0.4% of full scale and the output is stable 25μsec after the eight bit binary data is loaded into the appropriate latch. The latches are loaded by the write pulse (WR) whenever the proper address is presented to the MP-10. The lower two addresses (A₀ and A₁) are used internally by the device. Addresses A₂ & A₃ are compared with the address determination inputs B₂ and B₃. If their signals are found to be equal, and if addresses A₄-A₁₃ are all high, then the device is selected and one of the latches will be loaded. Address bit A₁ selects between output 1 and output 2. If address bit A₀ is set then the initialization channel of the DIA is selected. In order to prepare for operation a data pattern of 80H must

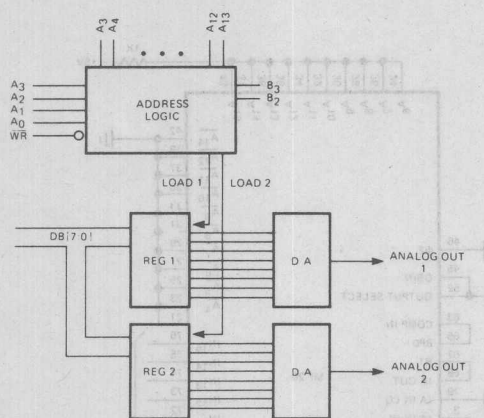


Figure 4. MP-10 Block Diagram

be output to this channel following the reset of the device.

A block diagram of the MP-20 analog to digital converter is shown in figure 5. This unit consists of a 16 input analog multiplexer, an instrumentation amplifier, an eight bit successive approximation analog to digital converter, and control logic. The 16 input multiplexer can be used to input either 16 single ended or 8 differential inputs. The output from the multiplexer is fed into the instrumentation amplifier which is configured so that it can easily be strapped for single ended 0-5 volt inputs, single ended ± 5 volt inputs, or differential 0-5 volt signals. Provisions are made for an external gain control resistor on the amplifier. The gain control equation is:

$$G = 2 + \frac{50k\Omega}{R_{ext}}$$

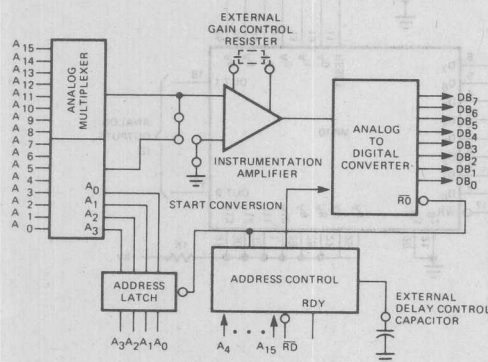


Figure 5. MP-20 Analog Subsystem

With no R_{ext} ($R_{ext} = \infty$) the gain is two and the input is 0-5 or ± 5 volts full scale. Adding an external resistor results in higher gain so that low level ($\pm 50mV$) signals from thermocouples and strain gauges can be accommodated. The output from the amplifier is applied to the actual A/D converter which provides an eight bit output with guaranteed monotonicity and an accuracy of $\pm 0.4\%$ of full scale. Note that this accuracy is specified for the entire module, not just for the converter itself. The control logic monitors address lines A15 through A4 to determine when the address of the unit has been selected. An address that the unit will respond to is determined by 11 address control pins, labeled $\overline{A4}$ through $\overline{A14}$. If one of these pins is tied to a logic 0 then the corresponding address pin must be high in order for the unit to be selected. If the pin is tied to a logic 1 then the corresponding address pin must be low. If the address of the module is selected when \overline{MEMR} pulse occurs, the lower four addresses ($A3-A0$) are stored in a latch which addresses the multiplexer. The coincidence of the proper address and \overline{MEMR} also initiates a conversion and gates the output of the converter on to the eight bit data bus.

The control logic of the MP-20 was designed to operate directly with an MCS-80TM system. When a \overline{MEMR} occurs and a conversion is initiated the MP-20 generates a READY signal which is used to extend the cycle of the 8080A for the duration of the conversion. READY is brought high after the conversion is complete which allows the 8080A to initiate a conversion and read the resulting data in a single, albeit long, memory or I/O cycle. The conversion time of the MP-20 depends on the gain selected for the amplifier. With no external resistor ($R = \infty$) the gain is two and the conversion time is 35 μsec . For $R = 510\Omega$ the gain is:

$$G = 2 + \frac{50k\Omega}{.51k\Omega} \approx 100$$

and the conversion time becomes 100 μsec . These settling times are specified in the MP-20 data sheet and range from 35 to 175 microseconds. The READY timing is controlled by an external capacitor. For a gain of 2 no external capacitor is required but if higher gains are selected a capacitor is needed to extend the timing.

A schematic showing both the MP-10 D/A and the MP-20 A/D connected to the 8748 is shown in Figure 6. This configuration, which consists of only four major components, gives an excellent example of what modern technology can do for

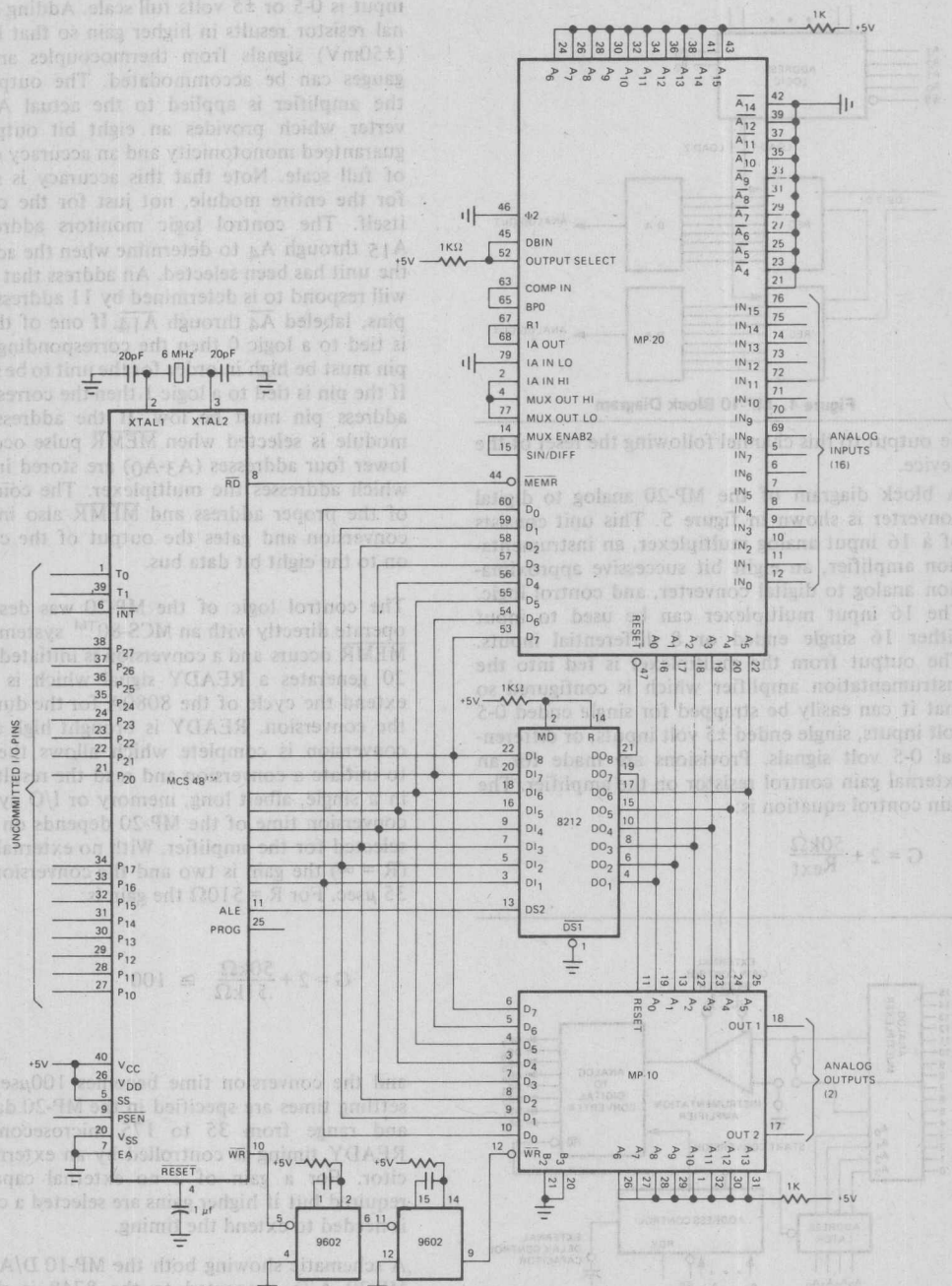
With no R_{EXT} ($R_{EXT} = \infty$) the gain is two and the input is 0.5 or 5.5 volts full scale. Adding an external resistor results in higher gain or full scale input (1.50mV) signals from thermocouples and strain gauges can be accommodated. The output from the amplifier is applied to the actual A/D converter which provides an eight bit output with guaranteed monotonicity and an accuracy of $\pm 0.4\%$ of full scale. Note that this accuracy is specified for the entire module, not just for the converter itself. The control logic monitors address lines A15 through A4 to determine when the address of the unit has been selected. An address that the unit will respond to is determined by 11 address control pins labeled A4 through A15. If one of these pins is used to a logic 0, the corresponding address pin must be high. If the pin is high, the corresponding address pin must be high. The address of the module is selected when MCLR pulse occurs. The lower four address (A3-A0) are stored in a latch which addresses the multiplier. The coincidence of the proper address and MCLR also initiates a conversion and takes the output of the converter to the eight bit data bus.

The control logic of the MP-20 is designed to operate directly with an MCS-48 system. When an MCLR occurs and a conversion is initiated the MP-20 generates a READY signal. The READY signal is used to extend the cycle of the 8080A microprocessor after the conversion. READY is complete when the 8080A is ready to read the data. The READY signal is active low. The gain of the MP-20 depends on the gain control equation:

$$G = 2 + \frac{50K\Omega}{R_{EXT}}$$

The conversion time of the MP-20 is 100µsec. These values are specified in the MP-20 data sheet and range from 35 to 100µsec. The external capacitor is selected as a capacitor. The gain control equation is selected as a capacitor.

Figure 6 shows both the MP-10 D/A and the MP-20 A/D connected to the 8748. This configuration, which consists of major components, gives an excellent example of what modern technology can do for



MCS-48™ Based Analog Processor

the system designer. The four components provide:

- An eight bit microprocessor
- 64 bytes of RAM
- 1024 bytes of UV erasable PROM
- A timer/event counter
- 16 digital I/O pins
- 2 testable input pins
- An interrupt capability
- 16 eight bit analog inputs
- 2 eight bit analog outputs

The MCS-48 communicates with the D/A and A/D converters in a memory mapped mode (i.e., it treats the devices as if they were external RAM). By setting an address in either R0 or R1 and then executing a MOVX the software can transfer data between the accumulator and the analog I/O. When the MCS-48 executes the MOVX instruction it first sends the eight bit address out on the bus and strobes it into the 8212 latch with the ALE (Address Latch Enable) signal. After the address is latched, the MCS-48 uses the same bus to transfer data to or from the accumulator. If data is being sent out (MOVX ∂R_j , A) the \overline{WR} strobe is used; if the data is being moved into the accumulator (MOVX A, ∂R_j) the \overline{RD} strobe is used. The one shots on the \overline{WR} line are used to delay the write strobe of the MCS-48 to meet the data set up specifications of the MP-10.

In order to provide reset capability for the analog devices without dedicating an I/O pin from the MCS-48, special addresses are used as reset channels. Executing any MOVX with an address of 0XXXXXXX will reset the A/D module; a similar operation with an address of X1XXXXXX will reset the D/A; a MOVX with an address of 01XXXXXX will reset both devices. All data transfers are accomplished with the upper two bits of the address field equal to 10. A summary of the addressing of the analog devices is shown in Table 1. Notice that except for an initialization channel for the D/A (which must

Table 1. Analog Interface Addresses

INPUT OR OUTPUT		
0 X X X	X X X X	Reset A/D
X 1 X X	X X X X	Reset D/A
INPUT		
0 0 1 1	n n n n	Read A/D Channel n n n n
OUTPUT		
1 0 1 1	0 0 0 1	Initialize D/A
1 0 1 1	0 0 0 0	Write Channel 1
1 0 1 1	0 0 1 0	Write Channel 2

All mnemonics copyrighted © Intel Corporation 1976.

be written to following a reset to initialize its internal logic) all channels involve some form of data transfer.

As was mentioned previously, the MP-20 was designed to use the READY line of the 8080A. Obviously this presents a problem since the MCS-48 does not support a READY line (with its attendant requirement of entering WAIT state). The necessity of a READY input can be overcome by performing a read operation to set the channel address, waiting the required delay (35 μ sec for a gain of two) and then performing a second read to actually obtain the data. The second read will read in the data from the channel selected by the first read irrespective of the channel selected for the second read. Thus it is possible to use the second read to set up the channel for the third read. Each read can read in the current channel and select the next channel for conversion.

The MP-20 is shown in Figure 6 strapped to input 16 single ended ± 5 volts signals. Programs which were used to test this configuration are shown in Figure 7. The first of these programs uses the D/A converter to generate sawtooth waveforms by outputting an incrementing value to the D/A converters. The second program scans the analog inputs and stores their digital values in a table located in RAM.

TABLE LOOKUP TECHNIQUES

```

LOC OBJ      SEQ      SOURCE STATEMENT
1          1          1          1          1
2          2          2          2          2
3          3          3          3          3
4          4          4          4          4
5          5          5          5          5
6          6          6          6          6
7          7          7          7          7
8          8          8          8          8
9          9          9          9          9
10         10         10         10         10
11         11         11         11         11
12         12         12         12         12
13         13         13         13         13
14         14         14         14         14
15         15         15         15         15
16         16         16         16         16
17         17         17         17         17
18         18         18         18         18
19         19         19         19         19
20         20         20         20         20
21         21         21         21         21
22         22         22         22         22
23         23         23         23         23
24         24         24         24         24
25         25         25         25         25
26         26         26         26         26
27         27         27         27         27
28         28         28         28         28
29         29         29         29         29
30         30         30         30         30
31         31         31         31         31
32         32         32         32         32
33         33         33         33         33
34         34         34         34         34
35         35         35         35         35
36         36         36         36         36
37         37         37         37         37
38         38         38         38         38
39         39         39         39         39
40         40         40         40         40
41         41         41         41         41
42         42         42         42         42
43         43         43         43         43
44         44         44         44         44
45         45         45         45         45
46         46         46         46         46
47         47         47         47         47
48         48         48         48         48
49         49         49         49         49
50         50         50         50         50
51         51         51         51         51
52         52         52         52         52
53         53         53         53         53
54         54         54         54         54
55         55         55         55         55
56         56         56         56         56
57         57         57         57         57
58         58         58         58         58
59         59         59         59         59
60         60         60         60         60
61         61         61         61         61
62         62         62         62         62
63         63         63         63         63
64         64         64         64         64
65         65         65         65         65
66         66         66         66         66
67         67         67         67         67
68         68         68         68         68
69         69         69         69         69
70         70         70         70         70
71         71         71         71         71
72         72         72         72         72
73         73         73         73         73
74         74         74         74         74
75         75         75         75         75
76         76         76         76         76
77         77         77         77         77
78         78         78         78         78
79         79         79         79         79
80         80         80         80         80
81         81         81         81         81
82         82         82         82         82
83         83         83         83         83
84         84         84         84         84
85         85         85         85         85
86         86         86         86         86
87         87         87         87         87
88         88         88         88         88
89         89         89         89         89
90         90         90         90         90
91         91         91         91         91
92         92         92         92         92
93         93         93         93         93
94         94         94         94         94
95         95         95         95         95
96         96         96         96         96
97         97         97         97         97
98         98         98         98         98
99         99         99         99         99
100        100        100        100        100
101        101        101        101        101
102        102        102        102        102
103        103        103        103        103
104        104        104        104        104
105        105        105        105        105
106        106        106        106        106
107        107        107        107        107
108        108        108        108        108
109        109        109        109        109
110        110        110        110        110
111        111        111        111        111
112        112        112        112        112
113        113        113        113        113
114        114        114        114        114
115        115        115        115        115
116        116        116        116        116
117        117        117        117        117
118        118        118        118        118
119        119        119        119        119
120        120        120        120        120
121        121        121        121        121
122        122        122        122        122
123        123        123        123        123
124        124        124        124        124
125        125        125        125        125
126        126        126        126        126
127        127        127        127        127
128        128        128        128        128
129        129        129        129        129
130        130        130        130        130
131        131        131        131        131
132        132        132        132        132
133        133        133        133        133
134        134        134        134        134
135        135        135        135        135
136        136        136        136        136
137        137        137        137        137
138        138        138        138        138
139        139        139        139        139
140        140        140        140        140
141        141        141        141        141
142        142        142        142        142
143        143        143        143        143
144        144        144        144        144
145        145        145        145        145
146        146        146        146        146
147        147        147        147        147
148        148        148        148        148
149        149        149        149        149
150        150        150        150        150
151        151        151        151        151
152        152        152        152        152
153        153        153        153        153
154        154        154        154        154
155        155        155        155        155
156        156        156        156        156
157        157        157        157        157
158        158        158        158        158
159        159        159        159        159
160        160        160        160        160
161        161        161        161        161
162        162        162        162        162
163        163        163        163        163
164        164        164        164        164
165        165        165        165        165
166        166        166        166        166
167        167        167        167        167
168        168        168        168        168
169        169        169        169        169
170        170        170        170        170
171        171        171        171        171
172        172        172        172        172
173        173        173        173        173
174        174        174        174        174
175        175        175        175        175
176        176        176        176        176
177        177        177        177        177
178        178        178        178        178
179        179        179        179        179
180        180        180        180        180
181        181        181        181        181
182        182        182        182        182
183        183        183        183        183
184        184        184        184        184
185        185        185        185        185
186        186        186        186        186
187        187        187        187        187
188        188        188        188        188
189        189        189        189        189
190        190        190        190        190
191        191        191        191        191
192        192        192        192        192
193        193        193        193        193
194        194        194        194        194
195        195        195        195        195
196        196        196        196        196
197        197        197        197        197
198        198        198        198        198
199        199        199        199        199
200        200        200        200        200
201        201        201        201        201
202        202        202        202        202
203        203        203        203        203
204        204        204        204        204
205        205        205        205        205
206        206        206        206        206
207        207        207        207        207
208        208        208        208        208
209        209        209        209        209
210        210        210        210        210
211        211        211        211        211
212        212        212        212        212
213        213        213        213        213
214        214        214        214        214
215        215        215        215        215
216        216        216        216        216
217        217        217        217        217
218        218        218        218        218
219        219        219        219        219
220        220        220        220        220
221        221        221        221        221
222        222        222        222        222
223        223        223        223        223
224        224        224        224        224
225        225        225        225        225
226        226        226        226        226
227        227        227        227        227
228        228        228        228        228
229        229        229        229        229
230        230        230        230        230
231        231        231        231        231
232        232        232        232        232
233        233        233        233        233
234        234        234        234        234
235        235        235        235        235
236        236        236        236        236
237        237        237        237        237
238        238        238        238        238
239        239        239        239        239
240        240        240        240        240
241        241        241        241        241
242        242        242        242        242
243        243        243        243        243
244        244        244        244        244
245        245        245        245        245
246        246        246        246        246
247        247        247        247        247
248        248        248        248        248
249        249        249        249        249
250        250        250        250        250
251        251        251        251        251
252        252        252        252        252
253        253        253        253        253
254        254        254        254        254
255        255        255        255        255
256        256        256        256        256
257        257        257        257        257
258        258        258        258        258
259        259        259        259        259
260        260        260        260        260
261        261        261        261        261
262        262        262        262        262
263        263        263        263        263
264        264        264        264        264
265        265        265        265        265
266        266        266        266        266
267        267        267        267        267
268        268        268        268        268
269        269        269        269        269
270        270        270        270        270
271        271        271        271        271
272        272        272        272        272
273        273        273        273        273
274        274        274        274        274
275        275        275        275        275
276        276        276        276        276
277        277        277        277        277
278        278        278        278        278
279        279        279        279        279
280        280        280        280        280
281        281        281        281        281
282        282        282        282        282
283        283        283        283        283
284        284        284        284        284
285        285        285        285        285
286        286        286        286        286
287        287        287        287        287
288        288        288        288        288
289        289        289        289        289
290        290        290        290        290
291        291        291        291        291
292        292        292        292        292
293        293        293        293        293
294        294        294        294        294
295        295        295        295        295
296        296        296        296        296
297        297        297        297        297
298        298        298        298        298
299        299        299        299        299
300        300        300        300        300
301        301        301        301        301
302        302        302        302        302
303        303        303        303        303
304        304        304        304        304
305        305        305        305        305
306        306        306        306        306
307        307        307        307        307
308        308        308        308        308
309        309        309        309        309
310        310        310        310        310
311        311        311        311        311
312        312        312        312        312
313        313        313        313        313
314        314        314        314        314
315        315        315        315        315
316        316        316        316        316
317        317        317        317        317
318        318        318        318        318
319        319        319        319        319
320        320        320        320        320
321        321        321        321        321
322        322        322        322        322
323        323        323        323        323
324        324        324        324        324
325        325        325        325        325
326        326        326        326        326
327        327        327        327        327
328        328        328        328        328
329        329        329        329        329
330        330        330        330        330
331        331        331        331        331
332        332        332        332        332
333        333        333        333        333
334        334        334        334        334
335        335        335        335        335
336        336        336        336        336
337        337        337        337        337
338        338        338        338        338
339        339        339        339        339
340        340        340        340        340
341        341        341        341        341
342        342        342        342        342
343        343        343        343        343
344        344        344        344        344
345        345        345        345        345
346        346        346        346        346
347        347        347        347        347
348        348        348        348        348
349        349        349        349        349
350        350        350        350        350
351        351        351        351        351
352        352        352        352        352
353        353        353        353        353
354        354        354        354        354
355        355        355        355        355
356        356        356        356        356
357        357        357        357        357
358        358        358        358        358
359        359        359        359        359
360        360        360        360        360
361        361        361        361        361
362        362        362        362        362
363        363        363        363        363
364        364        364        364        364
365        365        365        365        365
366        366        366        366        366
367        367        367        367        367
368        368        368        368        368
369        369        369        369        369
370        370        370        370        370
371        371        371        371        371
372        372        372        372        372
373        373        373        373        373
374        374        374        374        374
375        375        375        375        375
376        376        376        376        376
377        377        377        377        377
378        378        378        378        378
379        379        379        379        379
380        380        380        380        380
381        381        381        381        381
382        382        382        382        382
383        383        383        383        383
384        384        384        384        384
385        385        385        385        385
386        386        386        386        386
387        387        387        387        387
388        388        388        388        388
389        389        389        389        389
390        390        390        390        390
391        391        391        391        391
392        392        392        392        392
393        393        393        393        393
394        394        394        394        394
395        395        395        395        395
396        396        396        396        396
397        397        397        397        397
398        398        398        398        398
399        399        399        399        399
400        400        400        400        400
401        401        401        401        401
402        402        402        402        402
403        403        403        403        403
404        404        404        404        404
405        405        405        405        405
406        406        406        406        406
407        407        407        407        407
408        408        408        408        408
409        409        409        409        409
410        410        410        410        410
411        411        411        411        411
412        412        412        412        412
413        413        413        413        413
414        414        414        414        414
415        415        415        415        415
416        416        416        416        416
417        417        417        417        417
418        418        418        418        418
419        419        419        419        419
420        420        420        420        420
421        421        421        421        421
422        422        422        422        422
423        423        423        423        423
424        424        424        424        424
425        425        425        425        425
426        426        426        426        426
427        427        427        427        427
428        428        428        428        428
429        429        429        429        429
430        430        430        430        430
431        431        431        431        431
432        432        432        432        432
433        433        433        433        433
434        434        434        434        434
435        435        435        435        435
436        436        436        436        436
437        437        437        437        437
438        438        438        438        438
439        439        439        439        439
440        440        440        440        440
441        441        441        441        441
442        442        442        442        442
443        443        443        443        443
444        444        444        444        444
445        445        445        445        445
446        446        446        446        446
447        447        447        447        447
448        448        448        448        448
449        449        449        449        449
450        450        450        450        450
451        451        451        451        451
452        452        452        452        452
453        453        453        453        453
454        454        454        454        454
455        455        455        455        455
456        456        456        456        456
457        457        457        457        457
458        458        458        458        458
459        459        459        459        459
460        460        460        460        460
461        461        461        461        461
462        462        462        462        462
463        463        463        463        463
464        464        464        464        464
465        465        465        465        465
466        466        466        466        466
467        467        467        467        467
468        468        468        468        468
469        469        469        469        469
470        470        470        470        470
471        471        471        471        47
```

```

LOC OBJ SEQ SOURCE STATEMENT
1
2
3 TEST PROGRAM FOR ANALOG INPUT
4 THIS PROGRAM SCANS THE INPUT CHANNELS
5 AND STORES THE READINGS IN A TABLE
6 STARTING AT BUFF.
7
8
9
10 EQUATES
11
12
13 BUFF EQU 28H ; START OF BUFFER
14 MAXCH EQU 15 ; NO OF ANALOG INPUTS
15 AINCH EQU 88H ; BASE ADDRESS OF ANALOG INPUTS
16 TICK EQU 5 ; EXECUTION TIME OF DJNZ
17
18
19 START OF TEST
20
21
22 ORG 100H ; SETUP TO SCAN ANALOG INPUTS
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
0100 B92F 23 START: MOV R1, #BUFF-MAXCH
0102 B8BF 24 MOV R3, #MAXCH
0104 B8BF 25 MOV R8, #A(AINCH-MAXCH)
0106 08 26 MOVX A, @R8 ; SELECT CHANNEL 15
0107 BC8B 27 MOV R4, #48/TICK ; WAIT 48 MICROSECONDS
0109 EC89 28 DJNZ R4, $ ; NOW SCAN ANALOGS
010B C8 29 LOOP: DEC R8 ; GET DATA
010C 08 30 MOVX A, @R8 ; MOVE INTO BUFFER
010D A1 31 MOV @R1, A ; DECREMENT BUFFER POINT
010E C9 32 DEC R1
010F BC84 33 MOV R4, #28/MICROSEC ; PAD 28 MICROSEC
0111 EC11 34 DJNZ R4, $
0113 E8BB 35 DJNZ R3, LOOP ; LOOP UNTIL DONE
0115 2400 36 JMP START ; REPEAT TEST FOREVER
END 37 END OF PROGRAM

```

Figure 7b. A/D Exercise Program

TABLE LOOKUP TECHNIQUES

In the previous section the interface between analog I/O devices and the MCS-48™ was discussed. In many applications involving analog I/O one quickly finds that nature is inherently nonlinear, and the mathematics involved in 'linearizing it' can tax the computational power of the microprocessor, particularly if it has other tasks to perform. Problems of this nature are good candidates for the use of tables.

As an example of how tables can be used as part of an analog output scheme, consider a system which requires an MCS-48 to output a variable frequency sinusoidal waveform. One method of performing this function would be to use the timer to generate an interrupt at a fixed rate of 256 times the desired output frequency. At each interrupt the appropriate value of the sine function could be calculated from the MacLaurin series:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \dots \frac{(-1)^k x^{2k+1}}{(2K+1)!}$$

Where K is chosen to be large enough to provide the required accuracy.

All mnemonics copyrighted © Intel Corporation 1976.

The above calculation, although conceptually simple, would be time consuming and would severely limit the possible output frequencies which could be obtained. As an alternative to calculating these values in real time, the values could be precalculated off line and stored in a table. Upon each interrupt the MCS-48 would merely have to retrieve the appropriate value from the table and output it to the D/A converter. The MCS-48 provides a special instruction which can be used to access data in a table. If the table is stored in the last 256 bytes of the first kilobyte of MCS-48 memory then the table lookup can be performed by loading the independent variable (time in this case) into the accumulator and executing the instruction.

MOV3 A, @A

This instruction uses the initial contents of the accumulator to index into page 3 of program storage. The location pointed to is read and the contents placed in the accumulator. If (as is often the case) a table of fewer than 256 entries is required, then the table can be located in any page of program memory and the instruction:

MOV3 A, @A

can be used to retrieve data from the table. This instruction operates in the same manner as does the previous instruction except that the current page of program storage is assumed to contain the table.

If it is possible to devote slightly more of the microprocessor's time to the table look up process, then a much smaller table can often be utilized by taking advantage of interpolation to determine values of the function between values which are actual entries in the table. As an example of this

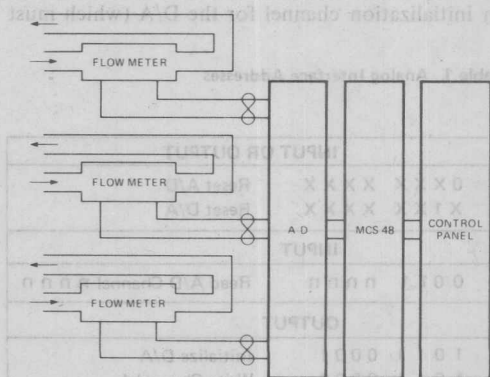


Figure 8. Flow Monitoring System

process consider the hypothetical system shown in Figure 8. The purpose of this system is to measure the flow through the three pipes, add them, and display the total flow on the control panel. The system consists of three flow meters which generate a differential voltage which is some function of flow, an A/D system with at least three differential inputs, an MCS-48, and a control panel. The schematic shown in Figure 6 could easily become part of this system, with the spare digital I/O of the MCS-48 used as an interface to the control panel. The simplicity of this system is clouded by the flow transducers, which are assumed to be not only nonlinear but also to require individual calibration (this is not an unreasonable assumption for a flow transducer). By using a table look up process and an 8748 the flow transducers can be calibrated and the results of the calibration tests stored directly in tables in the 8748. (The 8748 has a PROM in place of the ROM of the 8048 and thus makes such 'one off' programming practical.)

The results which might be obtained from calibrating one of the flow meters is shown in Figure 9. The results are plotted as gals/hour versus the measured voltage generated by the transducer. The voltage is shown in hexadecimal form so that it corresponds directly to the digital output of the analog to digital converter. The flow required to generate seventeen evenly spaced voltages (00H-100H in steps of 10H) has been measured and plotted. This information is shown in tabular form in Figure 10. It is necessary to generate a program which will convert any measured input from 00H to FFH into the flow in units which can be interpreted by a human operator. This can easily be done by simple interpolation.

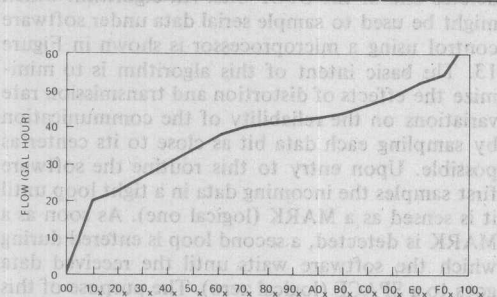


Figure 9. Flow Calibration Curve

Transducer Voltage (HEX)	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	100
Measured Flow (GAL/HOUR)	0	10	22	26	30	34	38	40	41	42	43	45	48	49	50	51	52

Figure 10. Tabulated Flow Data

The eight bits of independent variable (voltage) can be looked on as two four bit fields. The most significant four bits (7-4) will be used to retrieve one of the table values. The lower four bits (3-0) will be used to interpolate between this value and the value retrieved from the next higher location in the table. If the upper four bits are given the symbol I and the lower four bits the symbol N, then the interpolation can be expressed as:

$$F(x) = F(I) + \frac{N}{16} [F(I+1) - F(I)]$$

Where x is the measured voltage and F(x) is the corresponding flow.

If, as an example, the transducer voltage was measured as 48H then the flow (ref. Figure 10) would be:

$$F = 30 + \frac{8}{16} (34 - 30) = 32$$

A subroutine which implements this calculation is shown in Figure 11. Before it is called the independent variable (V) is placed in the accumulator and register R1 is set to point at the first value in the table. Aside from simple additions and subtractions the only arithmetic required is to multiply two values and then divide them by 16. The multiplication is handled via a subroutine which is also shown in Figure 11. The division by 16 can be performed by a four place right shift followed by a rounding operation. The routine shown will handle a monotonic increasing function of a single independent variable. Fairly simple modifications are required for nonmonotonic functions. Functions of two variables can be handled by interpolating on a plane rather than along a straight line. Although this is more time consuming, requiring an interpolation for each of the independent variables and a third to interpolate the final answer, it still provides a simple means of quickly calculating the required function. The use of tables can offer a powerful technique for function evaluation to the designer.

RECEIVING SERIAL CODE—BASIC APPROACHES

Many microprocessor based systems require some form of serial communication. Serial communication is extensively used because it allows two or more pieces of equipment to exchange information with a minimal number of interconnecting wires. The minimization of interconnecting wires results in simpler, cheaper, interconnects because fewer (or smaller) cables and connectors are required. Since the required number of drivers and receivers required is reduced, it can become economically feasible to provide much higher noise immunity

LOC	OBJ	SEQ	SOURCE STATEMENT	LOC	OBJ	SEQ	SOURCE STATEMENT
		1	*****	111C 83		56	RET
		2	*****			57	
		3	AT ENTRY R1 POINTS AT TABLE			58	
		4	A HAS INDEPENDENT VARIABLE			59	
		5	*****			60	MULTIPLY
		6	*****			61	
		7	*****			62	
		8	*****	111D 8800	63	MULT:	MOV COUNT, #8
		9	EQUATES	111F 8A00	64	MOV	AEX, #8
		10	*****			65	
		11	*****	1121 97	66	LOOPA:	CLR C
		12	RXB EQU R0 : POINTER #			67	
0000		13	RX1 EQU R1 : POINTER1	1122 122B	68	LOOPB:	JBH SSUM
0001		14	AEX EQU R2 : EXTENSION OF A REGISTER	1124 2A	69	XCH	A, AEX
0002		15	COUNT EQU R3 : COUNTER	1125 67	70	RRC	A
0003		16	TEMP EQU R4 : TEMP STORAGE	1126 2A	71	XCH	A, AEX
0004		17		1127 67	72	RRC	A
		18	*****			73	
		19	APPROXIMATION	1128 EB22	74	DJNZ	COUNT, LOOPB
		20		112A 83	75	RET	
		21				76	
0100		22	ORG 100H			77	SSUM: XCH A, AEX
		23	: POINT RXB AT TEMP	112C 68	78	ADD	A, @RXB
0100 8004		24	APPROX: MOV RXB, #TEMP	112D 67	79	RRC	A
		25	: TEMP-N AND BFH	112E 2A	80	XCH	A, AEX
		26	: A-P AND BFH	112F 67	81	RRC	A
0102 8000		27	MOV @RXB, #8			82	
0104 38		28	XCHD A, @RXB	1138 EB21	83	DJNZ	COUNT, LOOPA
0105 47		29	SWAP A	1132 63	84	RET	
		30				85	
0106 69		31	ADD A, RX1			86	
0107 A9		32	MOV RX1, A			87	
		33				88	TABLE TO TEST PROGRAM
		34	: RX1-TABLE(P)			89	*****
0108 E3		35	MOV P, A, @A			90	
0109 29		36	XCH A, RX1			91	ORG 300H
010A 17		37	INC A			92	
010B E3		38	MOV P, A, @A	0308 88	93	TABLE:	DB 88
		39		0301 8A	94		18
		40	: A-TABLE (P+1)-TABLE(P)	0302 16	95	DB	22
010C 37		41	CPL A	0303 1A	96	DB	26
010D 69		42	ADD A, RX1	0304 15	97	DB	38
010E 37		43	CPL A	0305 22	98	DB	34
		44		0306 26	99	DB	38
010F 341D		45	CALL MULT	0307 28	100	DB	48
0111 8002		46	MOV RXB, #AEX	0308 29	101	DB	41
0113 38		47	XCHD A, @RXB	0309 2A	102	DB	42
0114 47		48	SWAP A	030A 28	103	DB	43
0115 2A		49	XCH A, AEX	030B 2D	104	DB	45
0116 7219		50	JB3 ADJUST	030C 38	105	DB	48
0118 2A		51	XCH A, AEX	030D 31	106	DB	49
0119 2A		52	ADJUST: XCH A, AEX	030E 35	107	DB	53
011A 17		53	INC A	030F 38	108	DB	56
		54		0309 3F	109	DB	63
011B 69		55	ADD A, RX1		110		
		56			111	END	
		57	: RETURN				

Figure 11. Table Lookup With Interpolation

with more sophisticated (and expensive) line terminators. The final, and usually most persuasive, argument in favor of serial communication is that it may be the only method available to accomplish the job. The obvious example of this is telecommunications where it is necessary to encode parallel information into serial format in order to communicate via the telephone network. The intent of this section is to show how the facilities of the MCS-48™ can be brought to bear on the problem of serial communication.

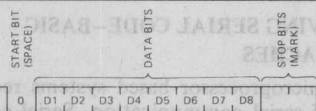


Figure 12. Serial ASCII Code

Probably the most common form of serial communication is that used by the ubiquitous Teletype-serial ASCII. This format, shown in Figure 12, consists of a START bit (0 or SPACE) followed by eight data bits which are in turn followed by two STOP bits (1 or MARK). In actual practice the

eight data bit usually consists of even parity on the remaining seven data bits; for the purposes of this discussion the eighth bit will be considered only as data. A minor variation of this format deletes one of the STOP bits. An algorithm which might be used to sample serial data under software control using a microprocessor is shown in Figure 13. The basic intent of this algorithm is to minimize the effects of distortion and transmission rate variations on the reliability of the communication by sampling each data bit as close to its center as possible. Upon entry to this routine the software first samples the incoming data in a tight loop until it is sensed as a MARK (logical one). As soon as a MARK is detected, a second loop is entered during which the software waits until the received data goes to a SPACE (logical zero). The purpose of this construction is to detect as accurately as possible the leading edge of the START bit. This instant of time will be used as a reference point for sampling all of the following bits in the character. After sensing the leading edge of the START bit a wait of one half the expected bit time is implemented. The period of the incoming signal is called P for convenience. At the end of this wait the serial line is tested—if it is MARK then the START bit was

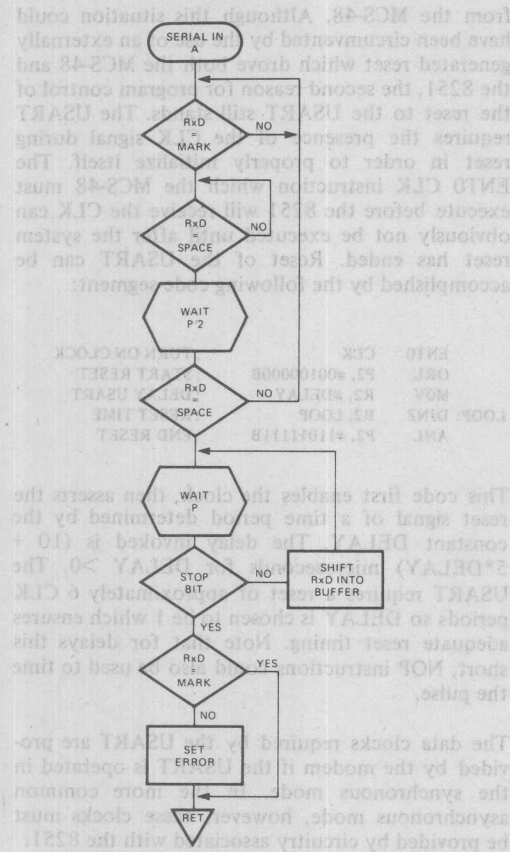


Figure 13. Sample Serial Input Routine

invalid and the process is reinitialized. If the line is still a SPACE, then the START bit is assumed to be valid and a delay of one bit time is started. At the completion of the delay the first data bit is sampled and a new delay of one bit time is initiated. This process is repeated until all eight data bits have been sampled. The last bit sampled is checked to determine if it is a valid STOP bit (a MARK). If it is, the character is assumed to be valid; if it is not, the character has a framing error and is probably invalid. A listing of a program which implements the above procedure is shown in Figure 14.

A disadvantage of the approach outlined in Figure 13 is that while the processor is inputting data serially it must totally dedicate itself to this task. Accurate timing can only be maintained if the program remains in a tight wait loop without allowing itself to be diverted to other functions. During reception of a character from a Teletype

the processor will spend only a 100μsecs or so processing data and the rest of the 100 milliseconds waiting to do the processing at the right time. This lack of efficiency (approximately 0.1%) in the utilization of processing power is why devices such as the 8251 USART find broad application in micro-processor systems.

LOC	OBJ	SEQ	SOURCE STATEMENT
0		1	*****
1		2	SIMPLE SERIAL INPUT
2		3	-THIS CODE ASSUMES RxD IS
3		4	CONNECTED TO PIN T8
4		5	*****
5		6	
6		7	
7		8	-----
8		9	EQUATES
9		10	-----
10		11	
0002		12	COUNT EQU R2 ; COUNTER
0000		13	BITNO EQU 8 ; NO OF BITS TO RECEIVE
0002		14	DLYH1 EQU 2 ; HI DLY COUNT
00A4		15	DLYLO EQU 0A4H ; LO DLY COUNT
0100		17	ORG 100H
0100 2600		18	19 SERIN: JNB T8 S ; LOOP UNTIL RxD=MARK
0102 3602		20	21 JTB S ; NOW LOOP UNTIL RxD=SPACE
0104 341C		22	23 CALL HBIT ; WAIT 1/2 BIT TIME
0106 3600		24	25 JTB SERIN ; IF FALSE START REINITIALIZE
0108 BA09		26	27 MOV COUNT, #BITNO+1 ; ELSE SET BIT COUNT
010A 341C		28	29 CALL HBIT ; WAIT 1 BIT TIME
010C 341C		30	31 CALL HBIT ; DECREMENT COUNT
		32	33 ; -IF ZERO EXIT WITH CARRY SET ON
		34	35 ; -FRAMING ERROR
010E EA15		36	37 DJNZ COUNT, LOAD
0110 97		38	39 CLR C
0111 3614		40	41 JTB EXIT
0113 A7		42	43 CPL C
0114 03		44	45 RET ; LOAD DATA
0115 97		46	47 CLR C
0116 2619		48	49 JNB LLLA
0118 A7		50	51 CPL C
0119 67		52	53 LLLA: RRC A
011A 240A		54	55 JMP LOOP ; AND LOOP
		56	57
		58	59
		60	END

Figure 14. Simple Serial Input

The 8251 USART is simple to interface to the MSC-48. Figure 15 shows such an interface. The USART requires a high speed clock (CLK), an initialization signal (RESET), data clocks (TxC and RxC), and data in order to operate. A circuit showing the connection of an 8748 to an 8251 USART is shown in Figure 15. In the circuit shown the high speed clock (which is used for internal sequencing by the USART) is provided by con-

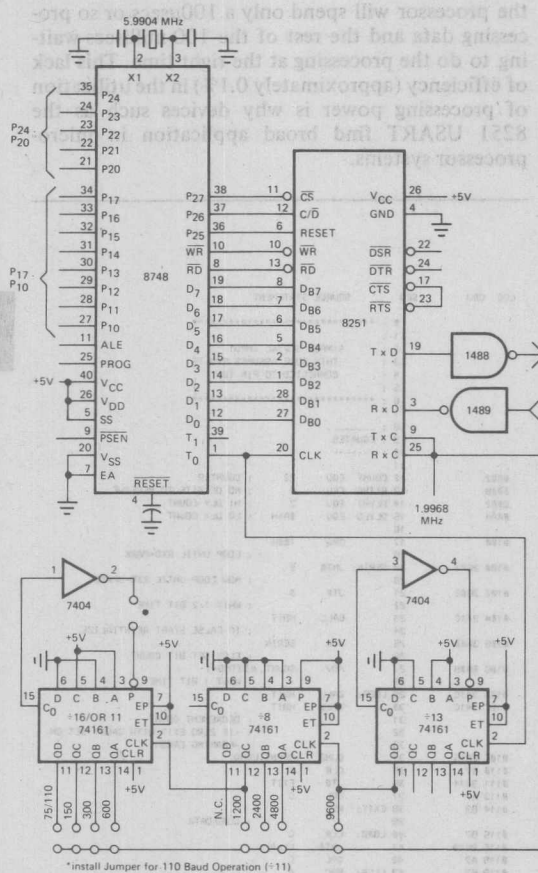


Figure 15. MCS-48™ to 8251 Interface

necting the CLK signal of the USART to the T₀ pin of the MCS-48. The T₀ pin of the MCS-48 can either be used as a directly testable input pin or it can become, under program control, an output pin which oscillates at one third of the crystal frequency. (Note that once this pin is designated by the software to be an output it will remain so until the system is reset.) In Figure 15 the crystal frequency is 5.9904 MHz so the clock provided to the 8251 is 1.9968 MHz, which conforms to its specifications.

The initialization signal to the USART (RESET) is provided programmatically by manipulation of bit 5 of port 2. It was necessary to place the reset of the 8251 under program control for two reasons. The first reason is that the MCS-48 does not supply a reset signal to other devices. The reason for this is that it was felt to be more useful to provide another pin of I/O function instead of a RESET OUT signal

from the MCS-48. Although this situation could have been circumvented by the use of an externally generated reset which drove both the MCS-48 and the 8251, the second reason for program control of the reset to the USART still stands. The USART requires the presence of the CLK signal during reset in order to properly initialize itself. The ENT0 CLK instruction which the MCS-48 must execute before the 8251 will receive the CLK can obviously not be executed until after the system reset has ended. Reset of the USART can be accomplished by the following code segment:

```

ENT0  CLK           ;TURN ON CLOCK
ORL   P2, #00100000B ;START RESET
MOV   R2, #DELAY     ;DELAY USART
LOOP: DJNZ R2, LOOP   ;RESET TIME
ANL   P2, #11011111B ;END RESET

```

This code first enables the clock, then asserts the reset signal of a time period determined by the constant DELAY. The delay invoked is (10 + 5*DELAY) microseconds for DELAY > 0. The USART requires a reset of approximately 6 CLK periods so DELAY is chosen to be 1 which ensures adequate reset timing. Note that for delays this short, NOP instructions could also be used to time the pulse.

The data clocks required by the USART are provided by the modem if the USART is operated in the synchronous mode. In the more common asynchronous mode, however, these clocks must be provided by circuitry associated with the 8251.

The 5.9904 MHz crystal was chosen because the resulting 1.9968 MHz clock to the USART can be evenly divided to provide transmit and receive clocks to the USART. Assuming the USART is in the x16 mode (i.e. it requires data clocks 16 times the baud rate) the 1.9968 MHz signal can be divided by 13 to generate the proper clock rate for 9600 baud operation. This 9600 baud clock can be further divided to give 4800, 2400, 1200, 600, and 300 baud signals. The 1200 baud signal can be divided by 11 to give a 109.1 baud signal which is within 1% of the 110 baud required by Teletypes.

The MCS-48 communicates with the 8251 in a memory mapped mode (i.e. as if the 8251 were external RAM). The instructions available to do this are MOVX @Rj, A which stores the contents of the accumulator at the external RAM location addressed by Rj (j=0 or 1), and its complement, the MOVX A, @ Rj instruction which moves data from the external RAM into the accumulator. Since the MCS-48 multiplexes addresses and data on the same eight bit bus an external latch would be required in order to address the USART with

Figure 16. 8251 Test Program

All mnemonics copyrighted © Intel Corporation 1976.

1

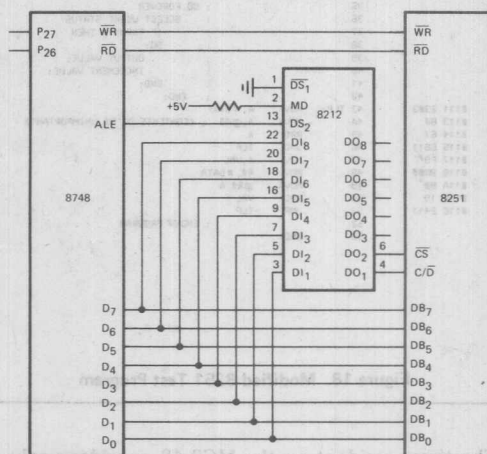


Figure 17. Modified MCS-48 to 8251 Interface

Although the USART does an admirable job of performing the serial I/O function for the MCS-48™, there are some situations where it can not be used. These situations may be caused by economic factors, such as an extremely cost sensitive design, or because the code which must be utilized cannot be accommodated by the USART. An example of such a code will be discussed later. Recall that the principal objection to the approach to serial input shown in Figure 13 was that it consumes much of the processor's power by merely spinning in loops in order to wait preset time delays.

LOC	OBJ	SEQ	SOURCE STATEMENT
1		1	SERIAL TEST
2		2	THIS CODE INITIALIZES THE USART
3		3	AND TRANSMITS AN INCREMENTING
4		4	PATTERN, HARDWARE SHOWN IF F10.17.
5		5	
6		6	
7		7	
8		8	EQUATES
11		11	MCLR EQU 20H ; USART RESET ADDRESS
12		12	DCY EQU 01H ; USART RESET DELAY
13		13	UCON EQU 03H ; USART CONTROL ADDRESS
14		14	MODE EQU 00EH ; USART MODE
15		15	CMD EQU 21H ; USART CMD
16		16	STAT EQU 03H ; USART STATUS
17		17	VAL EQU R1 ; TEST VALUE
18		18	DATA EQU 00 ; USART DATA ADDRESS
19		19	
20		20	
21		21	ORG 100H ; TURN ON CLOCK
22		22	AND RESET USART
23		23	TEST: ENTB CLK ;
24		24	ORL P2, #MCLR ;
25		25	MOV R2, #DLY ;
26		26	LOOP DJNZ R2, LOOP ;
27		27	ANL P2, #NOTMCLR ;
28		28	SELECT USART CONTROL
29		29	MOV A, #UCON ;
30		30	SEND MODE AND COMMAND
31		31	MOV A, #MODE ;
32		32	MOVX @R0, A ; (CONTENTS OF R0 UNIMPORTANT)
33		33	MOV A, #CMD ;
34		34	MOVX @R0, A ;
35		35	DO FOREVER
36		36	SELECT USART STATUS
37		37	IF TXRDY=1 THEN
38		38	DO ;
39		39	OUTPUT VALUE ;
40		40	INCREMENT VALUE ;
41		41	END ;
42		42	END ;
43		43	TLP: MOV A, #STAT ;
44		44	MOVX A, @R0 ; (CONTENTS OF R0 UNIMPORTANT)
45		45	RRC A ;
46		46	JNC TLP ;
47		47	MOV A, VAL ;
48		48	MOV R0, #DATA ;
49		49	MOVX @R0, A ;
50		50	INC VAL ;
51		51	TLP ;
52		52	JMP TLP ;
53		53	END ; END OF PROGRAM

Figure 18. Modified 8251 Test Program

The timer resident on the MCS-48 provides a solution to this problem. Instead of spinning in a loop the program can set the timer for a given interval, start it, and proceed to other tasks. When the timer overflows, an interrupt will be generated to notify the software that the present time period has elapsed. An extension of the algorithm of Figure 13 which uses the timer in this fashion is shown in Figure 19. This algorithm is identical to the preceding one up until the detection of the leading edge of the start bit. At this point the timer is set to one half of the bit time (P) and a return is made to the calling program which can start additional processing. At the completion of this time interval a timer overflow interrupt is generated. When the first interrupt is detected, the serial line is checked to ensure that it is in a spacing condition (valid START bit). If it is, the timer is set to P (to sample the middle of the first data bit) and a return is made to the program which was running when the

All mnemonics copyrighted © Intel Corporation 1976.

interrupt occurred. If the serial line has returned to the MARK state, a status flag is set to indicate an error and a return is made. On subsequent interrupt detection, the data is sampled, the timer is reinitiated, and control is returned to the program which was running when the interrupt occurred. When the last (i.e. STOP) bit is detected a completion flag is set and a return is made to the program running when the timer overflow occurred. By periodically checking the error and completion flags the running program can determine when the interrupt driven receive program has a character assembled for it.

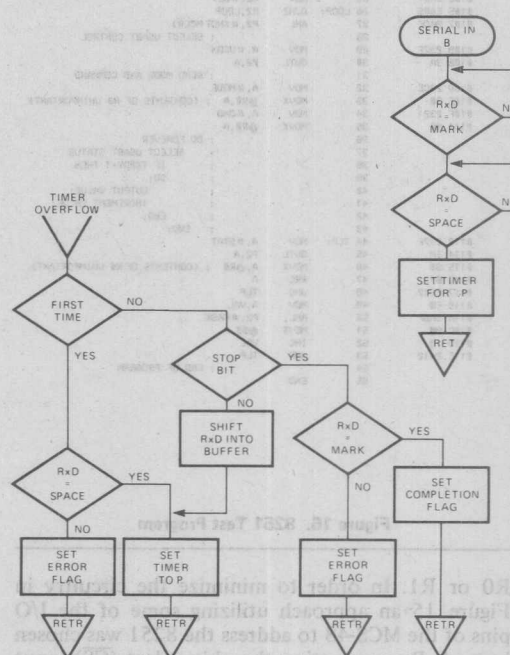


Figure 19. Improved Serial Input Routine

Using the timer to implement time delays as shown in Figure 19 results in considerable savings in processing time; two problems remain, however, which must be solved before an adequate software solution to the problem of receiving serial code can be found. The first problem is that even though the delays between bit samples are implemented via the timer rather than program loops the loop construction is still used to detect the leading edge of

the START bit. Although this results in the waste of processing power, the second problem is even more serious. For longer messages the required accuracy of the clocks becomes more and more stringent. Using the sampling technique discussed a cumulative error of one half a bit time in the time at which a bit sample is taken will result in erroneous reception. The maximum timing error which can be tolerated and yet still allow proper detection of an 11 bit ASCII character is then:

$$E_{max} = \frac{0.5 \cdot \text{BIT TIME}}{\text{CHARACTER TIME}} - \frac{0.5P}{11P} = 4.5\%$$

where P is the period of single bit. The corresponding calculation for a 32 bit character yields:

$$E_{max} = \frac{0.5P}{32P} = 1.6\%$$

Since this calculation does not allow for distortion on the signals, it is obvious that either extremely stable clocks will be required or a more tolerant algorithm must be devised. This problem is particularly serious at relatively high baud rates where the resolution of the counter (80μsec with a 6 MHz crystal) becomes a significant percentage of the period of the received signal. At the 110 baud rate of the Teletype the 80μsec resolution of the clock allows a maximum accuracy of 0.33%; at 2400 baud this figure is reduced to 3.8%.

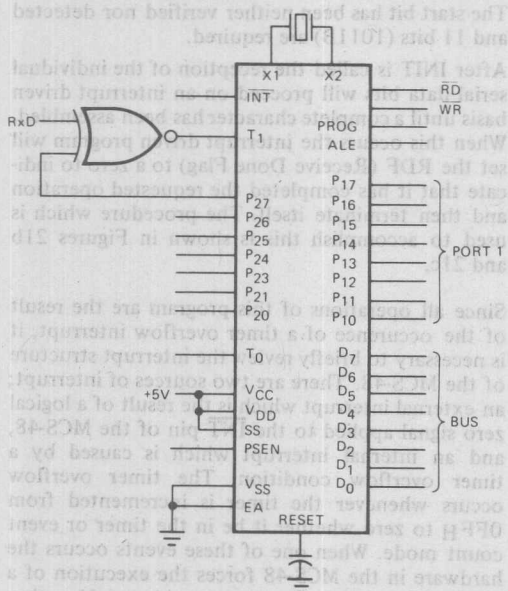


Figure 20. Detecting RxD Edges

Both efficient detection of the start bit and increased timing accuracy can be obtained if the MCS-48 can detect edges on the incoming received data (RxD). A hardware construct which allows this is shown in Figure 20.

The received data (RxD) is Exclusive NORed with bit seven of port two and fed into the TEST (T1) pin of the MCS-48. By manipulating P27 the program can now cause T1 to be either RxD or RxD. (If P27 = 1 then T1 = RxD; if P27 = 0 then T1 = RxD.) Note that not only can T1 be tested directly by the software but that it is the input which is used when the MCS-48 timer is in the event counter mode. The significance of this will be discussed later. The relationship between T1, P27, and RxD is given by the Boolean expression:

$$\overline{T1} = P27 \cdot \overline{RxD} + \overline{P27} \cdot RxD$$

Figure 21 flowcharts a means of utilizing this hardware construct to avoid the necessity of wasting time in program loops to detect the leading edge of the start bit. The receive operation is initialized when the program desiring to receive serial data calls the INIT subroutine (Figure 21a). Since INIT is going to manipulate the timer the first action it performs is to disable the timer overflow interrupt. Its next step is to set P27 to a logical 1. Setting P27 in this manner causes the TEST 1 input to the MCS-48 to follow RxD. By setting up the receive circuitry in this manner a high to low transition will occur on TEST 1 when the RxD goes from the MARKING to SPACING state (i.e. the START

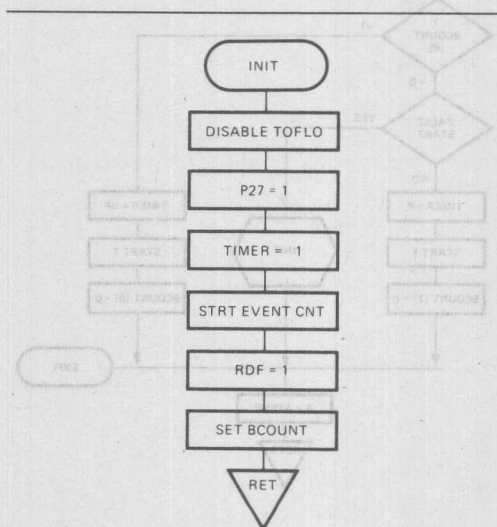


Figure 21a. Interrupt Driven Serial Receive Flowchart

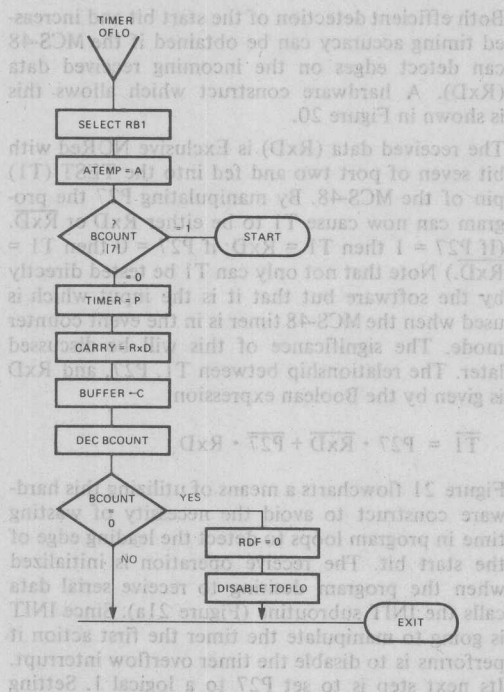


Figure 21b. Interrupt Driven Serial Receive Flowchart

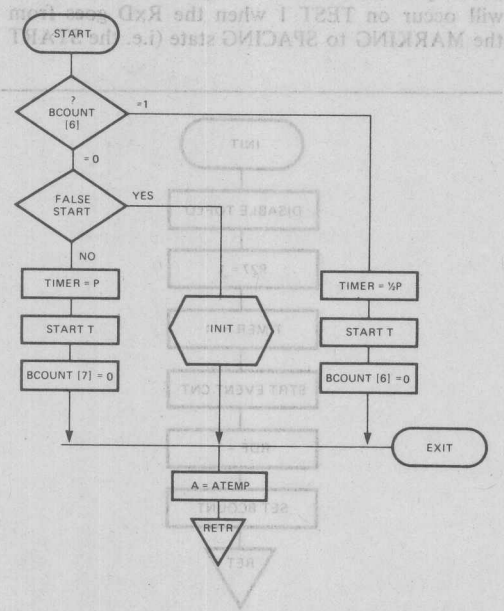
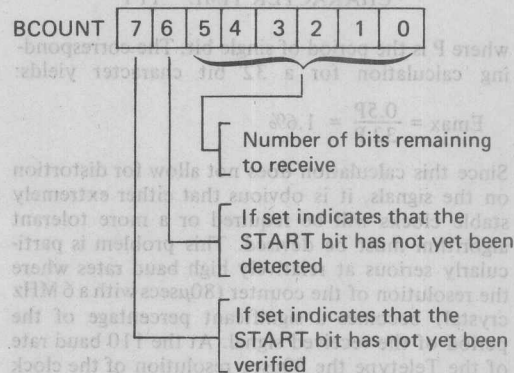


Figure 21c. Interrupt Driven Serial Receive Flowchart

bit occurs). By setting the timer to OFFH and enabling it in the event count mode, the INIT routine sets up the MCS-48 to generate a timer overflow interrupt on the next MARK to SPACE transition of RxD (the TEST 1 input doubles as the event counter input). Before returning to the calling program the INIT routine sets a flag (RDF) which will be cleared by the receive program when the requested receive operation is complete. INIT also sets a value into a register called BCOUNT. The receive program interprets BCOUNT as follows:



In order to request the reception of the 11 bit ASCII code INIT would set BCOUNT to 11001011B. The start bit has been neither verified nor detected and 11 bits (1011B) are required.

After INIT is called the reception of the individual serial data bits will proceed on an interrupt driven basis until a complete character has been assembled. When this occurs the interrupt driven program will set the RDF (Receive Done Flag) to a zero to indicate that it has completed the requested operation and then terminate itself. The procedure which is used to accomplish this is shown in Figures 21b and 21c.

Since all operations of this program are the result of the occurrence of a timer overflow interrupt, it is necessary to briefly review the interrupt structure of the MCS-48. There are two sources of interrupt; an external interrupt which is the result of a logical zero signal applied to the INT pin of the MCS-48, and an internal interrupt which is caused by a timer overflow condition. The timer overflow occurs whenever the timer is incremented from OFFH to zero whether it be in the timer or event count mode. When one of these events occurs the hardware in the MCS-48 forces the execution of a CALL. This CALL has a preset address of location 3 if it is due to the external interrupt and location 7 if it is due to a timer overflow. If both of these

events occur simultaneously the external interrupt will take precedence. The CALL automatically saves the contents of the program counter for the running program and its PSW (program status word) on a stack the hardware maintains in RAM locations 8-23. Although the hardware saves the program counter and PSW, it remains the responsibility of any interrupt driven software to make absolutely certain that it does not modify any memory locations or registers which are being used by the main program. The most convenient way of ensuring this in the MCS-48 is to dedicate the second bank of registers (RB1) to the interrupt driven program. One of these registers has to be used to save the accumulator (which is not part of the register bank) but seven registers remain; including two which can be used as pointers to the rest of the RAM (R0 and R1). Note that if this approach is taken then these registers have to be allocated between the program which services the external interrupt and the one which services the timer overflow. This problem is somewhat alleviated by a hardware lockout which prevents the timer overflow interrupt from interrupting the external interrupt service routine and vice versa. This is implemented by locking out new interrupts between the time an interrupt is recognized and the time a RETR instruction is executed. The RETR instruction is like a normal RET (return from subroutine) except that the PSW as well as the program counter is restored. The RETR instruction can be very much thought of as a return from interrupt instruction in the MCS-48.

The receive program under discussion uses register bank 1 in the manner described. Whenever a timer overflow occurs (e.g. on the next MARK to SPACE transition of RxD after INIT is called), control is passed (by the hardware generated CALL) to the point labeled TIMER OFLO in Figure 21b. This program segment immediately selects register bank 1 (RB1) and then saves the accumulator (A) in a location called ATEMP which is actually R7 of RB1. The program then tests bit seven of BCOUNT (R6 of RB1) to find out if a START bit has been verified (i.e. the edge of the START bit has first been detected and then verified to still be a SPACE one-half a bit time later. If BCOUNT [7] is a zero the START has been verified and the program proceeds to set the timer to P (the period of the serial bit), get the current serial data into the carry bit, and then shift the carry bit into a buffer. After saving the data the program decrements BCOUNT and tests it for zero. If BCOUNT is zero the receive operation is complete so the program sets RDF to a zero and disables timer overflow interrupts. Whether or not BCOUNT is zero, control is passed to EXIT where A is loaded with ATEMP and a

RETR is executed. Note that since the state of the flip flop which selects RB1 is saved as part of the PSW, the execution of RETR automatically selects the register bank which was active when the interrupt occurred.

If BCOUNT [7] is still set when it is tested, control is passed to START (Figure 21c) where bit 6 is tested to determine if the START has been detected yet. If BCOUNT [6] is set it indicates that this is the first occurrence of a timer overflow since the receive process was initialized by the INIT subroutine. If this is so, the program assumes that the START bit has just started and therefore it sets the timer to one-half of a bit time ($1/2 P$), starts the timer in the timer mode, and clears BCOUNT [6] to indicate that the START bit has been detected. The next overflow will again result in the execution of the program in Figure 21b and again BCOUNT [7] will be found to be set. This time, however, BCOUNT [6] will be reset and the program will know that it should test the START bit to ensure that it is still a SPACE. This test is performed and if successful the timer is set for a bit period P and BCOUNT [7] is reset so that on the next occurrence of a timer overflow the program will know that it should start assembling serial bits into a character. If the test is unsuccessful, the subroutine INIT is used to reinitialize the receive program. In either case control is passed to EXIT where a return from interrupt mode occurs.

This receive program, listings of which appear in Figure 22, allows the reception of serial characters transparently to the main running software. After INIT is called the main program has only to check RDF periodically to find out if there is data in the buffer for it. It would be fairly easy to 'double buffer' this operation by providing a buffer which the receive program uses to deserialize the incoming code and a second buffer to store the assembled character. If the program would reinitialize itself upon completion, the reception of a string of characters could proceed in much the same way as it would if a status driven USART were being used.

Although this program solves the first problem of software controlled reception (lack of efficiency) the second problem—sensitivity to frequency variations—remains. An example of a code which would be susceptible to this problem is the 31,26 BCH code commonly used in supervisory control systems. (A supervisory control system is, in essence, a remote control system which allows a human or computer operator the control of a system via a serial communications link.) The BCH codes are used because of their error detection capabilities and are a class of cyclical redundancy

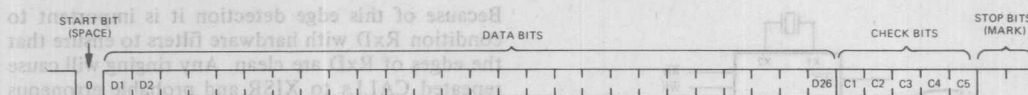


Figure 23. 31,26 BCH Code

A concept which reduces sensitivity to frequency deviations and thus allows the reception of longer codes is shown pictorially in Figure 24. The first line of this timing chart shows an alternative ones and zeros pattern on the RxD with a period of 5 milliseconds. The second line shows that by sampling at a period of exactly 5 milliseconds the data can be properly interpreted. The third and fourth lines show the effects of sampling with a period of six and four milliseconds respectively. In either case, an error occurs at the third sample where both periods result in sampling on an edge of the RxD signal. The third line of Figure 24 shows a hybrid sampling scheme which, based on some additional information, switches sampling periods between the two values. As can be seen in Figure 24, the data is sampled with a 4 millisecond period until the sampling begins to fall behind the data; at this point the sampling period is increased to six milliseconds and the sampling first catches up and then passes the center point of the data. As soon as this happens, the sampling period reverts to the 4 millisecond period and the cycle repeats. It can be seen that this scheme sets up a pattern which repeats indefinitely and the data can be successfully sampled. Note that the sampling pattern established is alternating periods of four and six milliseconds. The average period of this pattern, as might be expected, is 5 msec. Line 5 of Figure 24 shows the effect of a change in transmission speed to a period of 5.5 msec with no change in the sampling time. The sampling is again successful but the new sampling pattern is 4-6-6-6; 4-6-6-6, etc. Note that the average sample is again equal to the period of the received data (5.5). While this scheme

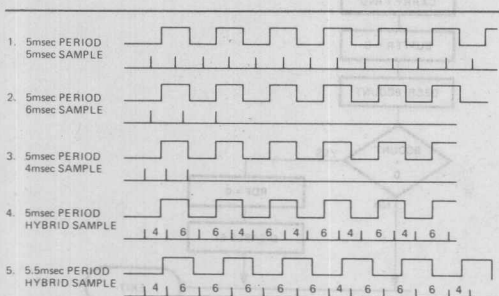


Figure 24. Various Sampling Alternatives

does seem to work, the question of what additional information is needed remains.

The MSC-48 must somehow decide when it is drifting out of synchronization and take corrective action. By referring back to Figure 24 it can be seen that if the MCS-48 could determine where the edges of RxD occurred with respect to its sampling times then the additional information would be available. As can be seen in the figure the choice of sampling period can be based on the following rule:

If an edge on the RxD line occurs during the first half of the current sampling period, then use the short period for the next sample. If an edge occurs during the second half of the period, then use the long sampling period for the next sample.

If the data on the RxD line does not change, of course, the MCS-48 will drift out of synchronization just as the original algorithm did. As long as edges occur on TxD, however, synchronization can be maintained. To maximize the allowable time between edges, the following addition could be made to the above rule:

If no edge occurs on the RxD line during a sample, then change sampling period from short to long or vice versa.

Note that this addition to the rule will result in using an average of the two sampling periods when no edge occurs for several bit times.

The edges of RxD can be easily detected by the use of the same structure (the Exclusive — NOR gate) which was added to the MCS-48 in Figure 20. This gate, which is used to detect the edge on RxD which begins the START bit, can naturally be used to detect any edge. Since the timer is being used to time the bit period, however, the event count input (T1) is not useful during the receive itself. By connecting the output of this gate, however, to the INT input to the MCS-48 (see Figure 25) it is possible to detect edges on RxD with the event counter when the program is trying to detect the START bit and by the external interrupt when the program is using the timer to control the sampling times.

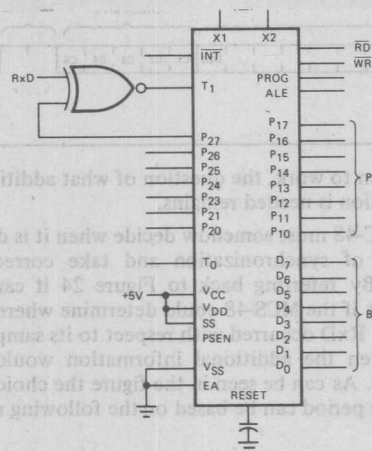
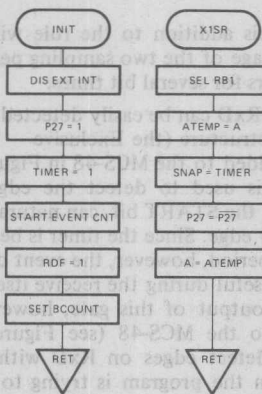


Figure 25. Modified Edge Detection

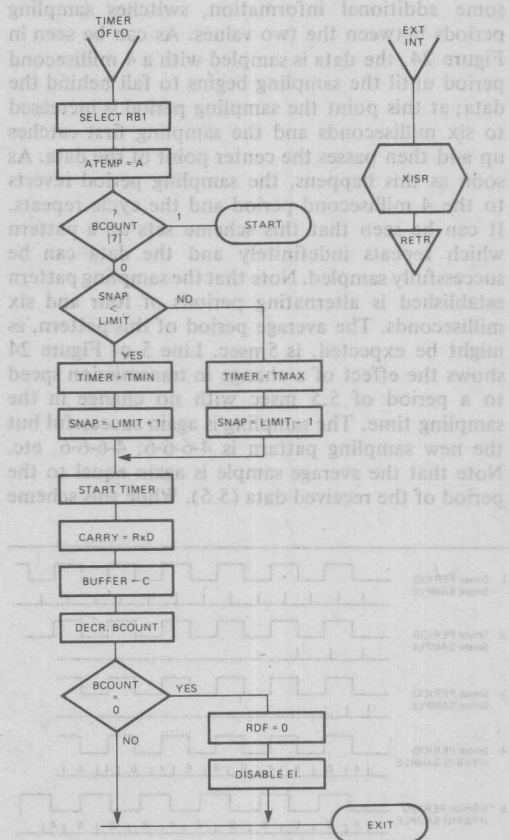
A modification to the program of Figure 21 which implements this new sampling algorithm is shown in Figure 26. The first deviation from the original program is the addition of a routine (XISR, Figure 26a) which is called when an external interrupt occurs (i.e. when an edge occurs on RxD). This routine saves the status of the running program and then stores the current value of the timer register in a location called SNAP (R5 of RB1). After doing these operations the program complements bit 7 of port 2. Manipulating P27 in this manner will cause the Exclusive NOR gate to turn off the external interrupt and will set it up to generate another interrupt when the RxD line changes again (has another edge).



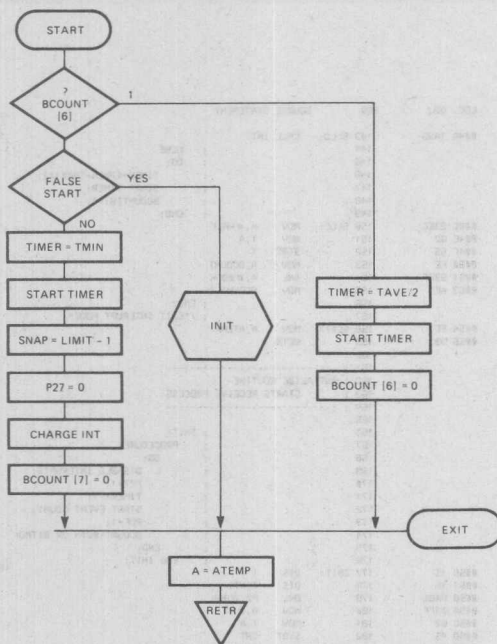
Hybrid Sampling Flowchart

condition KxD with hardware filters to ensure that the edges of RxD are clean. Any ringing will cause repeated CALLs to XISR and probable erroneous operation. The changes to the START process (Figure 26c) are two-fold; first the TIMER is set to one half the average of the two sample periods when the START bit is first detected (BCOUNT [6] = 1), and second the processing of the edge information is initialized by presetting SNAP and clearing P27.

SNAP is preset so that when the reception of data actually begins (Figure 26b BCOUNT [7] = 0), the decision block which tests SNAP against LIMIT will be initialized. This block actually compares the value in SNAP with a LIMIT value which is used to determine if the sampling point is ahead or behind the actual midpoint of the serial data. If the sampling is ahead then the timer is set for TMIN; if the sampling is behind then the timer is set for



Hybrid Sampling Flowchart



Hybrid Sampling Flowchart

TMAX. By presetting SNAP in the manner shown in the flowcharts the second rule of the algorithm, (if no edge appears on the RxD line during a sample, then change the sampling periods short to long or vice versa) is automatically met. If an edge occurs then XISR will modify SNAP, if XISR is not invoked between two samples then the choice of timer periods will alternate. The only other significant change to the algorithm is that the INIT routine must now lock out all interrupts, not just the timer overflow interrupt, while it is operating. A program which uses this algorithm to receive a 32 bit message is shown in Figure 27.

LOC	OBJ	SEG	SOURCE STATEMENT
8			1:
9			2:
10			3: SERIAL INPUT USING MCS-48
11			4: THIS CODE ASSUMES HARDWARE
12			5: SHOWN IN FIG 25. PROGRAM
13			6: IS SIMILAR TO PREVIOUS
14			7: ONE. A MORE SOPHISTICATED
15			8: SAMPLING ALGORITHM IS USED
16			9:
17			10: NOTE: A PL/M LIKE LANGUAGE WAS USED
18			11: TO COMMENT THIS LISTING AND
19			12: SEVERAL OTHERS IN THIS NOTE. NO
20			13: COMPILER EXISTS FOR THE MCS-48.
21			14: THE COMMENTS WERE 'HAND
22			15: COMPILED' INTO ASSEMBLY CODE
23			16:
24			17:
25			18:
26			19:
27			20: EQUATES
28			21:
29			22:
30			23: ATMP EQU R7 ; STORAGE FOR A DURING INTERRUPT
31			24: BCOUNT EQU R6 ; CONTAINS NUMBER OF BITS IN MSG
32			25: SNAP EQU R5 ; TAKES TIMER SNAP SHOT ON RxD EDGE
33			26: COUNT EQU R2 ; UTILITY COUNTER
34			27: RXD EQU R0 ; POINTER
35			28: BITNO EQU 32 ; NUMBER OF BITS
36			29: LIMIT EQU 28 ; TEST VALUE FOR MIN/MAX SAMPLING
37			30: TMAX EQU -43 ; MAX SAMPLE PERIOD
38			31: TMIN EQU -39 ; MINIMUM SAMPLE PERIOD
39			32: HALF EQU -28 ; HALF NOMINAL PERIOD
40			33: SERBUF EQU 20H ; START OF SERIAL BUFFER
41			34: RDP EQU 24H ; RECEIVE DONE FLAG
42			35:
43			36:
44			37: CONTROL PASSED HERE ON EXT. INT.
45			38:
46			39:
47			40: ORG #3H ; CALL SERVICE ROUTINE
48			41:
49			42: EIVEC: CALL XISR
50			43: RETR
51			44:
52			45:
53			46: CONTROL PASSED HERE WHEN TIMER OFLO OCCURS
54			47:
55			48:
56			49:
57			50:
58			51:
59			52:
60			53:
61			54:
62			55:
63			56:
64			57:
65			58:
66			59:
67			60:
68			61:
69			62:
70			63:
71			64:
72			65:
73			66:

Figure 27. Hybrid Sampling Program


```

LOC OBJ    SEQ    SOURCE STATEMENT
0019 62      74      MOV     T,A
001A BD13    75      MOV     SNAP,#LIMIT-1
001C 55      76      ; START TIMER;
001D 8A      77      SLB:   STRT    T
001E 77      78      ; /*CARRY-RXD*/
001F 4622    79      ; CARRY=P27 XOR TEST1;
0021 A7      80      IN      A,P2
0022 B828    81      RLC     A
0023 62      82      JNT1   TISR
0024 BA04    83      CPL     C
0025 62      84      ; /*SHIFT CARRY INTO BUFFER*/
0026 26      85      ; RXB=SERBUF;
0027 67      86      COUNT=1;
0028 28      87      ; DO WHILE COUNT<=8;
0029 10      88      ; RSHFT MEM(RXD);
002A EA26    89      ; RXB=RXB<1;
002B 82      90      COUNT=COUNT-1;
002C EES4    91      END;
002D 82      92      TISR:  MOV     RXB,#SERBUF
002E 82      93      MOV     COUNT,#4
002F 67      94      SLOOP: XCH     A,RXB
0030 28      95      RRC     A
0031 10      96      XCH     A,@RXB
0032 82      97      INC     RXB
0033 15      98      DJNZ    COUNT,SLOOP
0034 8454    99      ; BCOUNT=BCOUNT-1;
0035 62      100     ; IF BCOUNT=0 THEN
0036 FE      101     DJNZ    BCOUNT,SEXIT
0037 D24C    102     ; DO;
0038 82      103     ; RDX=0;
0039 82      104     ; DISABLE EX INT;
003A 82      105     ; END;
003B 82      106     MOV     RXB,#RDX
003C 27      107     CLR     A
003D 1A0      108     MOV     @RXB,A
003E 35      109     DIS     TONTI
003F 15      110     DIS     I
0040 8454    111     ; END;
0041 8454    112     JMP     EXIT
0042 8454    113     ; ELSE
0043 8454    114     ; DO;
0044 8454    115     ; IF BCOUNT(6)=0 THEN
0045 FE      116     START:  MOV     A,BCOUNT
0046 D24C    117     JBS     SLLC
0047 82      118     ; DO;
0048 82      119     ; IF TEST1=0 THEN
0049 82      120     ; DO;
0050 82      121     ; DO;
0051 82      122     ; TIMER=TMIN;
0052 82      123     ; START TIMER;
0053 82      124     ; SNAP=LIMIT-1;
0054 82      125     ; P27=0;
0055 82      126     ; EN I
0056 82      127     ; BCOUNT(7)=0;
0057 82      128     ; END;
0058 82      129     MOV     A,#TMIN
0059 82      130     MOV     T,A
0060 82      131     STRT    T
0061 82      132     MOV     SNAP,#LIMIT-1
0062 82      133     ANL     P2,#7FH
0063 82      134     CN      A,1
0064 82      135     MOV     A,BCOUNT
0065 82      136     ANL     A,#7FH
0066 82      137     MOV     BCOUNT,A
0067 82      138     JMP     EXIT
0068 82      139     ; ELSE
0069 82      140     ; DO;
0070 82      141     ; CALL INIT;
0071 82      142     ; END;

```

```

LOC OBJ    SEQ    SOURCE STATEMENT
004A 1456    143     SLLD:  CALL INIT
004B 1456    144     ; ELSE
004C 1456    145     ; DO;
004D 1456    146     ; TIMER=(TMIN-TMAX)/2;
004E 1456    147     ; START TIMER;
004F 1456    148     ; BCOUNT(6)=0;
0050 1456    149     ; END;
0051 23EC    150     SLLC:  MOV     A,#HALF
0052 62      151     MOV     T,A
0053 55      152     STRT    T
0054 FE      153     MOV     A,BCOUNT
0055 53BF    154     ANL     A,#BFH
0056 82      155     MOV     BCOUNT,A
0057 82      156     ; END;
0058 82      157     ; /*EXIT INTERRUPT MODE*/
0059 FF      158     SEXIT:  MOV     A,ATEMP
0060 93      159     RETR
0061 82      160     ;
0062 82      161     ; -----
0063 82      162     ; INITIALIZE ROUTINE-
0064 82      163     ; STARTS RECEIVE PROCESS
0065 82      164     ; -----
0066 82      165     ; INIT:
0067 82      166     ; PROCEDURE;
0068 82      167     ; DO;
0069 82      168     ; DISABLE INTERRUPTS;
0070 82      169     ; P27=1;
0071 82      170     ; TIMER=1;
0072 82      171     ; START EVENT COUNT;
0073 82      172     ; RDX=1;
0074 82      173     ; BCOUNT=BCBH OR BITNO
0075 82      174     ; END;
0076 82      175     ; END INIT;
0077 82      176     ;
0078 15      177     INIT:  DIS     I
0079 35      178     DIS     TONTI
0080 8454    179     ORL     P2,#BBH
0081 23FF    180     MOV     A,#-1
0082 62      181     MOV     T,A
0083 45      182     STRT    CNT
0084 82      183     MOV     RXB,#RDX
0085 82      184     MOV     A,B1
0086 1A0      185     MOV     @RXB,A
0087 25      186     EN      TONTI
0088 BEE8    187     MOV     BCOUNT,#BBBH OR BITNO
0089 82      188     RET
0090 82      189     ;
0091 82      190     ; -----
0092 82      191     ; INTERRUPT SERVICE ROUTINE
0093 82      192     ; -----
0094 82      193     ; XISR:
0095 82      194     ; PROCEDURE;
0096 82      195     ; DO;
0097 82      196     ; /*ENTER INTERRUPT MODE*/
0098 82      197     ; SNAP=TIMER;
0099 82      198     ; P27=NOT P27;
0100 82      199     ; END XISR;
0101 82      200     ;
0102 82      201     XISR:  SEL     RB1
0103 82      202     MOV     ATEMP,A
0104 82      203     MOV     A,T
0105 82      204     MOV     SNAP,A
0106 82      205     IN      A,P2
0107 82      206     XRL     A,#BBH
0108 82      207     OUTL    P2,A
0109 82      208     MOV     A,ATEMP
0110 82      209     RET
0111 82      210     ; END OF PROGRAM;
0112 82      211     END

```

TMAX by presc...
 in the flowchart...
 It no edge...
 the sampling...
 long or the...
 occur then...
 not invoked...
 of timer...
 significant...
 routine must...
 the timer...
 A program...
 32 bit message...

Figure 27. Hybrid Sampling Program

Figure 27. Hybrid Sampling Program

TRANSMITTING SERIAL CODE

Serial transmission is conceptually far simpler than serial reception since no synchronization is required. All that is required is to use the timer to generate interrupts at the bit rate and present the character to be transmitted serially at an I/O pin. A program which does this is shown in Figure 28. The transmission of serial data becomes much more complicated if it must occur simultaneously with reception.

If both reception and transmission are to occur simultaneously then obviously contention will exist for the use of the timer. It is possible to allow the simultaneous reception and transmission of serial data using the timer as a general clock which controls software maintained timers. The attainable baud rates using such techniques are, however, limited and the use of a 8251 USART is probably

indicated in all but the most cost sensitive applications. An exception to this rule occurs when the system, although full duplex in nature, actually transmits the same data as it receives. An example of this is a microprocessor driving a terminal such as a Teletype. Although the circuit to the terminal is full duplex, the data that is transmitted is generally the same as that received. A minor modification to the program shown in Figure 26 would implement this mode of operation. The modification would be to the XISR routine and it would add the code necessary to place the TxD I/O pin in the same state as the RxD line. Since any change in RxD results in a call to XISR, this modification would cause the retransmission of any received data. Whenever it becomes necessary to transmit data which is not being received, the program of Figure 28 could be used in a half duplex manner.

```

LOC OBJ    SEQ    SOURCE STATEMENT
0
1:
2: SERIAL TRANSMIT ON THE MCS48
3: TO USE PUT A CHAR IN BUFF AND
4: SET CHARAV TO 0FFH, WHEN THE
5: TRANSMITTER IS READY FOR ANOTHER
6: CHAR IT WILL CLEAR CHARAV, THE
7: TRANSMISSION IS DOUBLE BUFFERED.
8:
9:
10:
11: EQUATES
12:
13:
0007 14 ATEMP EQU R7 ; STORAGE FOR A DURING INT.
0008 15 PTOS EQU R6 ; PARALLEL TO SERIAL CONVERTER.
0009 16 BUFF EQU R5 ; CHARACTER BUFFER
000A 17 CHARAV EQU R4 ; CHARACTER AVAILABLE FLAG
000B 18 COUNT EQU R3 ; BIT COUNTER
000C 19 CBIT EQU 0FFH ; MASK TO CLEAR TxD IN P24
000D 20 SBIT EQU 010H ; MASK TO SET TxD IN P24
000E 21 P EQU -41 ; PERIOD OF TxD
000F 22
0010 23:
0011 24: CONTROL PASSED HERE ON TIMER OVERFLOW
0012 25:
0013 26 ORG 07H
0014 27
0015 28 TOFLO: SEL RB1 ; ENTER INTERRUPT MODE
0016 29 MOV ATEMP,A
0017 30 ; SET TIMER FOR P
0018 31 MOV A,#P
0019 32 MOV T,A
001A 33 STRT T
001B 34 ; GET BIT INTO CARRY
001C 35 CALL BIT
001D 36 ; SET TxD TO CARRY

```

```

LOC OBJ    SEQ    SOURCE STATEMENT
000F BA 37 IN A,P2
0010 B8 38 ARL A,#BBH
0011 B9 39 OUTL P2,A
0012 CA 40 JC BITON
0013 CB 41 ANL P2,#CBIT
0014 CC 42 JMP EXIT
0015 CD 43 BITON: ORL P2,#SBIT
0016 CE 44 EXIT: MOV A,ATEMP
0017 CF 45 RETR
0018 D0 46
0019 D1 47:
001A D2 48: BIT ROUTINE
001B D3 49: PICKS THE NEXT BIT TO TRANSMIT
001C D4 50:
001D D5 51
001E D6 52 BIT: MOV A,COUNT
001F D7 53 JZ IDLE
0020 D8 54 MOV A,PTOS
0021 D9 55 RRC A
0022 DA 56 ORL A,#BBH
0023 DB 57 MOV PTOS,A
0024 DC 58 DEC COUNT
0025 DD 59 RET
0026 DE 60
0027 DF 61 IDLE: CLR C
0028 E0 62 MOV A,CHARAV
0029 E1 63 JNZ GOTONE
002A E2 64 CPL C
002B E3 65 RET
002C E4 66
002D E5 67 GOTONE: MOV A,BUFF
002E E6 68 MOV PTOS,A
002F E7 69 MOV COUNT,#18
0030 E8 70 MOV CHARAV,#8
0031 E9 71 RET
0032 EA 72
0033 EB 73 END OF PROGRAM
0034 EC 73 END

```

Figure 28. Serial Transmission

GENERATING PARITY

Many communications schemes require the generation and checking of parity. If a USART is used it can be programmed to automatically generate and check parity. If the communications is handled by software within the MCS-48™ then the program must perform parity calculations. Calculating parity is easy if one remembers what parity really means. A character has even parity if the number of one bits in it is even. A character has odd parity if it has an odd number of ones. The program segment shown in Figure 29 can be caused to calculate parity. It starts by setting a loop count to eight and

```

LOC  OBJ  SEQ  SOURCE STATEMENT
      1
      2 : *****
      3 :
      4 : PARITY
      5 : THIS PROGRAM GENERATES PARITY
      6 : ON THE ACCUMULATOR
      7 : CARRY WILL BE SET IF A HAS ODD PARITY
      8 :
      9 : *****
     10 :
     11 :
     12 : -----
     13 : EQUATES
     14 : -----
     15 :
0002  16 COUNT EQU R2
     17 :
0100  18 PAR: ORG 100H
0100  19 MOV COUNT, #8 ; SET LOOP COUNT
0102  20 CLR C ; INITIALIZE CARRY
     21 : FOR EACH ZERO BIT IN A
     22 : COMPLEMENT THE CARRY FLAG
0103  22 LOOP: RR A
0104  23 JNB OVER ; IF ZERO, NO CARRY
0106  24 CPL C ; COMPLEMENT CARRY
     25 :
     26 :
     27 : END OF PROGRAM
     28 : END
  
```

Figure 29. Parity Generation

clearing the CARRY flag. After this initialization a loop is executed eight times. During each execution the accumulator is rotated and the least significant bit is tested. If the bit is a zero the CARRY flag is complemented, if the bit is a one no further action is taken. Since an even number of zeros implies an even number of ones for an eight bit character, after all eight loops have been accomplished the CARRY bit will be set if an odd number of ones were encountered; it will be reset if the number were even. Since the RR instruction does not involve CARRY the net result of executing this program loop is to set CARRY if parity is odd without effecting the character in the accumulator.

CONCLUSION

This Application Note has presented a very small sampling of the application techniques possible with the MCS-48™ family. The application of this new single chip computer system to tasks which have not yet yielded to the power of the micro-processor will present a fascinating challenge to the system designer.

would be greatly reduced.

Key

Since keyboard and display logic can't just one or several functions handled by a microprocessor, the added cost of including three functions in a system is minimal. In fact, considering the extremely low cost of X-Y matrix keyboards and monochrome displays—**June 1978**

then use it often more cost effectively than even a hardwired discrete switches and indicators. Thus, the additional flexibility of keyboard input and display output can be added to inexpensive computer products (such as games clocks, thermometers, tape recorders etc.) while producing a net savings to system cost.

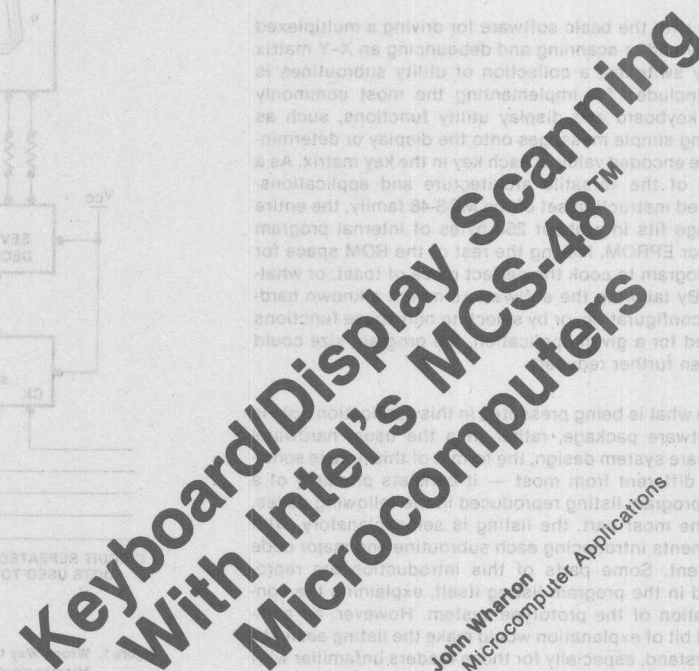
Display Scanning MCS-48™ Controllers

Intel's Microcomputer

John Wharton
Microcomputer Applications

1

June 1978



Keyboard/Display Scanning With Intel's MCS-48™ Microcomputers

John Wharton
Microcomputer Applications

John Wharton
Microcomputer A

INTRODUCTION

This application note presents a software package for interfacing members of Intel's MCS-48™ family of single-chip microcomputers with keyboards and displays using a minimum of external components. Because of the similarity of the architectures of the various members of the family (the 8035, 8048, 8748, 8039, 8049, 8021, and 8022 microcomputers; also the 8041 and 8741 universal peripheral interfaces in the UPI-41® family), the code included here could run with minor modifications on any member of the family.

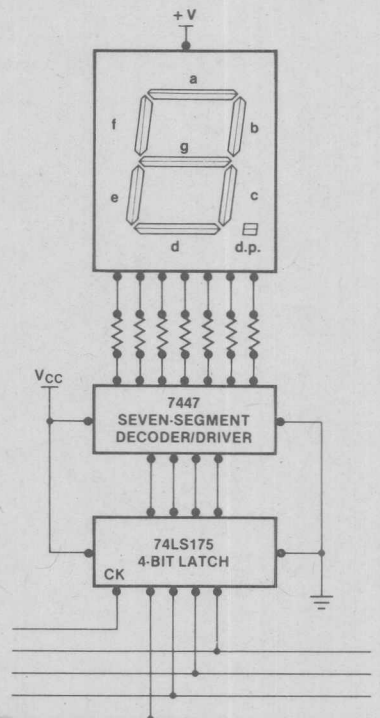
Since keyboard and display logic can be just one of several functions handled by a microprocessor, the added cost of including these functions in a system is minimal. In fact, considering the extremely low cost of standard X-Y matrix keyboards and integrated displays, their use is often more cost effective than even a handful of discrete switches and indicators. Thus, the additional flexibility of keyboard input and display output can be added to inexpensive consumer products (such as games, clocks, thermostats, tape recorders, etc.), while producing a net savings in system cost.

Since each potential application will have its own unique combination of keys and display characters, the program is written so that very little modification is needed to interface it with a wide variety of hardware configurations. In general, the only changes required are within the set of initial EQUates at the beginning of the program.

Along with the basic software for driving a multiplexed display and/or scanning and debouncing an X-Y matrix of key switches, a collection of utility subroutines is also included for implementing the most commonly used keyboard and display utility functions, such as copying simple messages onto the display or determining the encoded value of each key in the key matrix. As a result of the versatile architecture and applications-oriented instruction set of the MCS-48 family, the entire package fits into about 250 bytes of internal program ROM or EPROM, leaving the rest of the ROM space for the program to cook the perfect piece of toast, or whatever. By tailoring the software to match a known hardware configuration, or by selecting only those functions needed for a given application, the program size could be even further reduced.

Since what is being presented in this application note is a software package, rather than the usual hardware/software system design, the format of this note is somewhat different from most — it consists primarily of a long program listing reproduced in the following pages. For the most part, the listing is self-explanatory, with comments introducing each subroutine and major code segment. Some parts of this introduction are reproduced in the program listing itself, explaining the configuration of the prototype system. However, an additional bit of explanation would make the listing easier to understand, especially for those readers unfamiliar with the concept of multiplexed displays and keyboards.

In traditional digital system design, various hardware registers or counters were used to hold binary or BCD values which had to be conveyed to the user. The standard way of presenting this information was by connecting each register to a seven-segment encoder (such as the 7447) driving a single display character, as represented by Figure 1. Thus, two ICs, seven current limiting resistors, and about 45 solder joints were required for each digit of output. Consider how traditional techniques might be (mis-)applied in designing a microprocessor system: the designer could add a latch, encoder, and resistors for each digit of the display. Still another latch and decoder could be used to turn on one of the decimal points (if used). The characters displayed could only be a sequence of decimal digits. In the same vein, a large matrix of key switches could be read by installing an MSI TTL priority encoder read by an additional input port. Not only would all this use a lot of extra I/O ports and increase the system price and part count drastically, but the flexibility and reliability of the system would be greatly reduced.



CIRCUIT REPEATED FOR EVERY DIGIT OF DISPLAY
(DOTS USED TO INDICATE SOLDER JOINTS)

Figure 1. Wrong Way to Design Multiple Digit Displays for Microcomputer Systems

Instead, a scheme of time-multiplexing the display can be used to decrease costs, part count, and interconnections, while allowing a wider range of character types to be used on the display. The techniques used here are fairly typical of today's integrated subsystems designed especially for controlling keyboards and displays (such as in calculators or the Intel® 4269, 8278, and 8279 Keyboard/Display Controller Devices).

In a multiplexed display, all the segments of all the characters are interconnected in a regular two-dimensional array. One terminal of each segment is in common with the other segments of the same character; the other terminal is connected with the same segments of the other characters. This is represented schematically in Figure 2. A digit driver or segment driver is needed for each of these common lines.

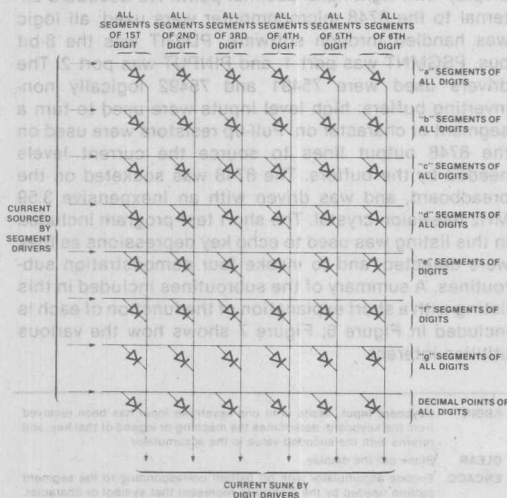


Figure 2. Schematic Representation of 6-Digit, 7-Segment Common-Cathode LED Multiplexed Display

The various characters of the display are not all on at once; rather, only one character at a time is energized. As each character is enabled, some combination of segment drivers is turned on, with the result that a digit appears on the enabled character. (For example, in Figure 3, if segment drivers 'a', 'b', and 'c' were on when character position #6 was enabled, the digit '7' would appear in the left-most place.) Each character is enabled in this way, in sequence, at a rate fast enough to ensure that the display characters seem to be on constantly, with no appearance of flashing or flickering.

In the system presented here, these rapid modifications to the display are all made under the control of the MCS-48™ microcomputer. At periodic intervals the computer quickly turns off all display segments, disables the character now being displayed and enables the next, looks up the pattern of segments for the next character

to be displayed, and turns on the appropriate segments. With the next character now turned on, the processor may now resume whatever it had been doing before. The whole display updating task consumes only a small fraction of the processor's time.

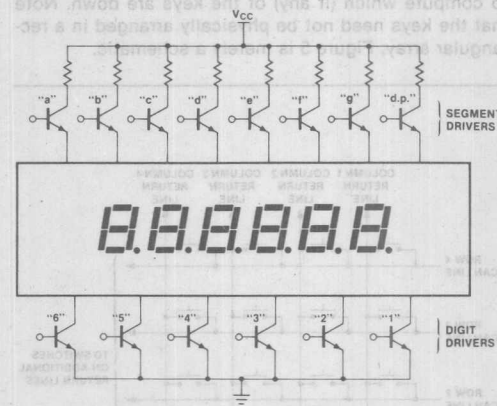


Figure 3. Segment and Digit Drivers used with 6-Position, 7-Segment LED Display

Moreover, since the computer rather than a standard decoder circuit is used to turn the segments off and on, patterns for characters other than decimal digits may be included in the display. Hexadecimal characters, special symbols, and many letters of the alphabet are possible. With sufficient imagination this feature can be exploited for some applications, as suggested by the examples in Figure 4.

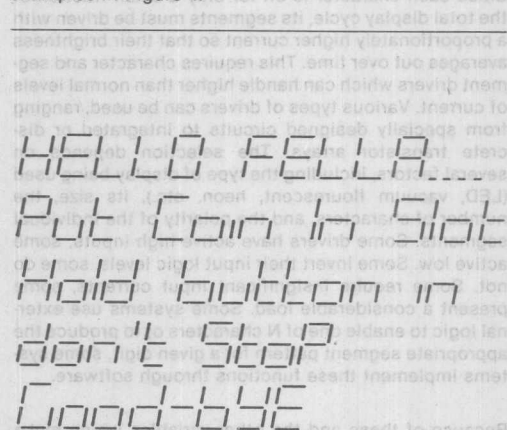


Figure 4. Examples of Typical Messages Possible with Simple 7-Segment Displays

As each character of the display is turned on, the same signal may be used to enable one row of the key matrix. Any keys in that row which are being pressed at the time will then pass the signal on to one of several "return lines", one corresponding to each column of the matrix. (See Figure 5.) By reading the state of these control lines, and knowing which row is enabled, it is possible to compute which (if any) of the keys are down. Note that the keys need not be physically arranged in a rectangular array; Figure 5 is merely a schematic.

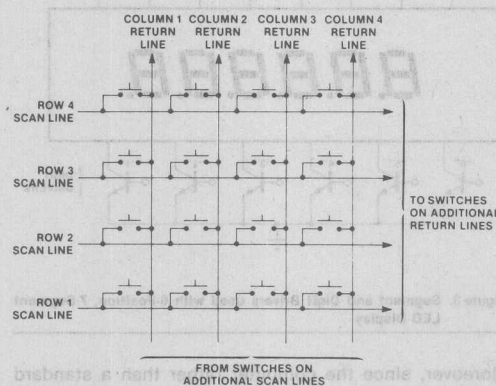


Figure 5. Schematic of X-Y Matrix Multiplexed Keyboard

Since each character is on for only a small fraction of the total display cycle, its segments must be driven with a proportionately higher current so that their brightness averages out over time. This requires character and segment drivers which can handle higher than normal levels of current. Various types of drivers can be used, ranging from specially designed circuits to integrated or discrete transistor arrays. The selection depends on several factors, including the type of display being used (LED, vacuum fluorescent, neon, etc.), its size, the number of characters, and the polarity of the individual segments. Some drivers have active high inputs, some active low. Some invert their input logic levels, some do not. Some require insignificant input currents, some present a considerable load. Some systems use external logic to enable one of N characters or to produce the appropriate segment pattern for a given digit, some systems implement these functions through software.

Because of these and the other variables which make each application unique, provisions are made in the first page of symbol EQUates to allow the user to specify such things as the number of characters in the display or the polarity of the drivers used, and the program will be assembled accordingly. The display is refreshed on each timer interrupt, which occurs every $32 \times (\text{TICK})$

machine cycles. (One machine cycle occurs every 30 crystal oscillations for the 8021 and 8022, or every 15 oscillations for all other members of the family.) A more detailed explanation of these variables is included in the listing.

Port assignment is also at the discretion of the user — all port references in the listing are "logical" rather than physical port names. The port used to specify which character is enabled is referred to as "PDIGIT". The output segment pattern is written to "PSGMNT" and the keyboard return lines are read by "PINPUT". These logical port names may be assigned to whichever ports the user pleases.

By way of example, the breadboard used to develop and debug this software used a matrix of 16 single-pole pushbuttons and an 8-character common-cathode LED display with right-hand decimal point. No decoders external to the 8748 microcomputer were used; all logic was handled through software. PDIGIT was the 8-bit bus, PSGMNT was port 1, and PINPUT was port 2. The drivers used were 75491 and 75492 logically non-inverting buffers; high level inputs were used to turn a segment or character on. Pull-up resistors were used on the 8748 output lines to source the current levels needed by the buffers. The 8748 was socketed on the breadboard, and was driven with an inexpensive 3.59 MHz television crystal. The short test program included in this listing was used to echo key depressions as they were detected, and to invoke four demonstration subroutines. A summary of the subroutines included in this listing with a short explanation of the function of each is included in Figure 6; Figure 7 shows how the various utilities interact.

KBDIN	Keyboard Input. Waits until one keystroke input has been received from the keyboard; determines the meaning or legend of that key, and returns with the encoded value in the accumulator.
CLEAR	Blank out the display.
ENCAACC	Encode accumulator with bit pattern corresponding to the segment pattern needed by the display to represent that symbol or character. Uses the value of the accumulator when called to access a table containing the patterns for all legal input values.
WDISP	Write into Display. Writes the bit pattern in the accumulator into the next character position of the display. Maintains a character position counter so that repeated calls will automatically write characters into sequential positions.
RENTRY	Right-hand Entry. Stores the accumulator segment pattern in the display in the right-most character position. Shifts all other characters to the left one place.
PRINT	Print a string of arbitrary characters onto the display. Useful for prompting messages, warnings, etc. Uses a table of segment patterns in ROM, so that messages will not be restricted to numbers, letters, etc.
FILL	Fill the display with the character pattern in the accumulator. Useful for writing dashes, segment test patterns, etc., into all character positions.
ECHO	Wait for a key to be pressed by the operator and write that key onto the display. Used for providing feedback to the operator when entering numeric data, etc.
RDPADD	Adds or deletes a decimal point to the character at the right-hand side of the display, for entering floating point numbers.
HOLD	Called when a key is known to be down. Does not return until all keys have been released. Used for organ-type keyboards, or when some action should not be initiated until the key invoking that action has been released.
DELAY	Provides a crude real-time delay corresponding to the value of the accumulator when called. Can be used to cause display characters to blink, to momentarily flash information, to enable a buzzer, etc. Could also be used by the program when delays are needed, such as to slow down the computer reaction rate while playing a game against the human operator.

Figure 6. Utility Subroutine Definitions

ISIS-II MCS-48/UP1-41 MACRO ASSEMBLER, V2.0
AP40 INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
1			\$MACROFILE XREF
2			\$TITLE 'AP40 INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX'
3			
4			THE FOLLOWING SOFTWARE PACKAGE PROVIDES A SEVEN SEGMENT DISPLAY
5			INTERFACE FOR MICROCOMPUTERS IN THE INTEL MCS-48 FAMILY
6			THE CODE IS WRITTEN SO THAT VARIOUS HARDWARE
7			CONFIGURATIONS CAN BE ACCOMMODATED BY REDEFINING THE INITIAL VARIABLES
8			IN MOST SITUATIONS, THE KEYBOARD/DISPLAY INTERFACE WILL BE REQUIRED TO
9			IMPLEMENT MORE SOPHISTICATED SINGLE-CHIP SYSTEMS (CALCULATORS, SCALES, CLOCKS,
10			ETC.), WITH SECTIONS OF THE FOLLOWING CODE SELECTED AND MODIFIED AS NECESSARY
11			FOR EACH APPLICATION
12			
13			A SINGLE SUBROUTINE (CALLED REFRESH) IS USED TO IMPLEMENT BOTH THE DISPLAY
14			MULTIPLEXING AND KEYBOARD SCANNING, USING THE SAME SIGNAL BOTH TO ENABLE
15			ONE CHARACTER OF THE DISPLAY AND TO STROBE ONE ROW OF THE X-Y KEY MATRIX
16			THE SUBROUTINE MUST BE CALLED SUFFICIENTLY OFTEN TO ENSURE THE DISPLAY
17			CHARACTERS DO NOT FLICKER- AT LEAST 50 COMPLETE DISPLAY SCANS PER SECOND
18			TO ACCOMMODATE SWITCHES OF ARBITRARY CHEAPNESS, THE DEBOUNCE TIME CAN BE
19			SET TO BE ANY DESIRED NUMBER OF COMPLETE SCANS
20			THUS THE DEBOUNCE TIME IS A FUNCTION OF BOTH THE SCAN RATE AND THE VALUE
21			OF CONSTANT 'DEBNC'
22			
23			IN THIS LISTING, THE INTERNAL TIMER IS USED TO GENERATE INTERRUPTS THAT
24			SERVE AS A TIME BASE FOR THE REFRESH SUBROUTINE
25			ALTERNATE TIME BASES MIGHT BE AN EXTERNAL OSCILLATOR (DRIVING THE INTERRUPT
26			PIN OR POLLED BY A TEST OR INPUT PIN), A SOFTWARE DELAY LOOP IN THE BACKGROUND
27			PROGRAM, OR PERIODIC CALLS TO THE SUBROUTINE FROM THROUGHOUT THE USER'S PROGRAM
28			AT APPROPRIATE PLACES
29			IN THESE CASES, THE CODE STARTING AT LABEL TIINT (TIMER INTERRUPT) AND TIRET
30			(TIINT RETURN) COULD STILL BE USED TO SAVE AND RESTORE ACCUMULATOR CONTENTS
31			THE INTERRUPT SERVICING ROUTINE SELECTS REGISTER BANK 1
32			FOR THE NEEDED REGISTERS
33			
34			
35			WRITTEN BY JOHN WHARTON, INTEL SINGLE-CHIP COMPUTER APPLICATIONS
36			
37			\$EJECT

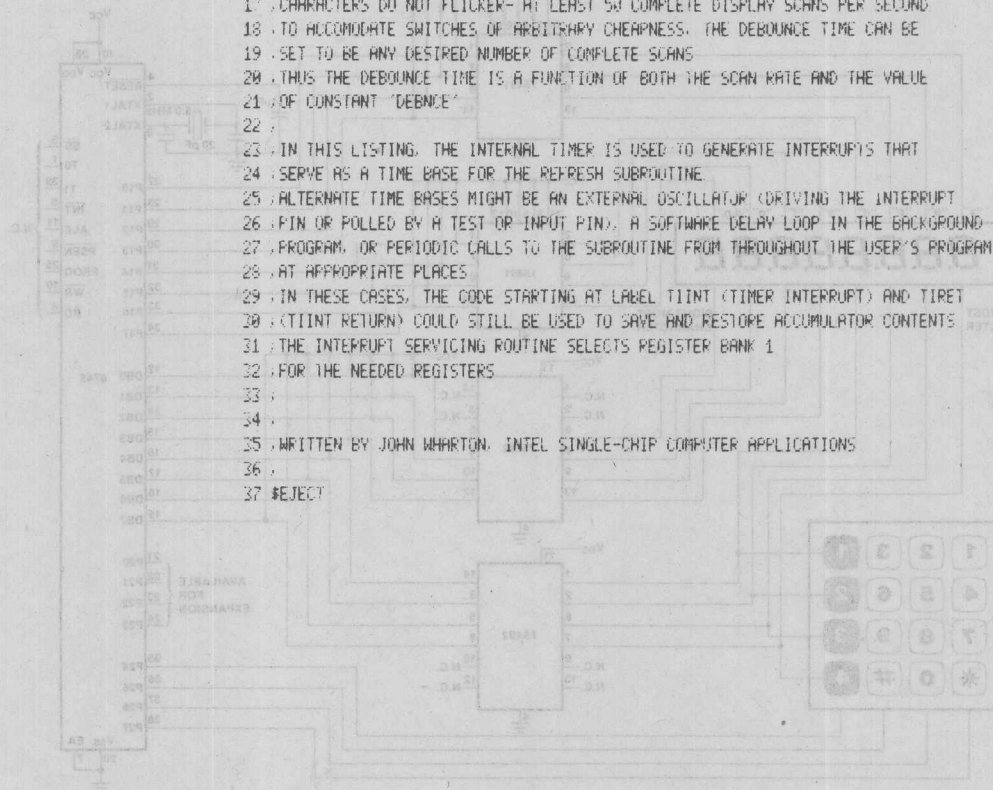


Figure 8 Intel MCS-48 Keyboard/Display Application

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT	TARGET STATEMENT	322	LOC
38			IN THIS IMPLEMENTATION OF THE DISPLAY SCAN, IT IS ASSUMED THAT THERE WILL			
39			BE RELATIVELY LITTLE I/O OTHER THAN FOR THE KEYBOARD/DISPLAY			
40			IF THIS IS THE CASE, THEN THERE IS NO NEED FOR ANY ADDITIONAL EXTERNAL			
41			LOGIC (SUCH AS ONE-OF-EIGHT DECODERS OR SEVEN-SEGMENT ENCODERS), THOUGH			
42			THERE WILL STILL BE A NEED FOR CURRENT OR VOLTAGE DRIVERS, ACCORDING TO			
43			THE TYPE OF DISPLAY BEING USED.			
44						
45			IN THIS LISTING, THE PROCESSOR I/O PORTS ARE LOGICALLY DIVIDED AS FOLLOWS			
46						
47			A DIGIT-EIGHT BIT PORT USED TO ENABLE, ONE AT A TIME, THE INDIVIDUAL			
48			CHARACTERS OF AN EIGHT DIGIT SEVEN-SEGMENT DISPLAY, WHILE ALSO			
49			STROBING THE ROWS OF AN X-Y MATRIX KEYBOARD.			
50			BIT7 ENABLES THE LEFTMOST CHARACTER AND THE BOTTOM ROW OF THE KBD.			
51			BIT4 ENABLES THE TOP ROW OF THE 4x4 KBD AND THE FOURTH CHARACTER.			
52			BIT0 ENABLES THE RIGHTMOST CHARACTER.			
53			(A 4x8 KEYBOARD COULD BE STROBED BY ALSO USING BIT3-BIT0			
54			AND EXTENDING OR ELIMINATING THE TABLE, "LEGENDS".)			
55			THE ENABLING OF ONE BIT (ACTIVE HIGH OR LOW) IS ACCOMMODATED BY			
56			ACCESSING A LOOK-UP TABLE CALLED CHSTB.			
57			THIS TECHNIQUE TAKES ABOUT FOUR BYTES MORE ROM THAN A TECHNIQUE			
58			OF ROTATING A 'ONE' THROUGH A FIELD OF 'ZEROS' IN THE ACC.			
59			AN APPROPRIATE NUMBER OF TIMES, BUT IT ALLOWS SOME ADDITIONAL			
60			FLEXIBILITY: IF THE DRIVERS BEING USED HAVE A COMBINATORIAL INPUT			
61			(AS IN THE 7545X FAMILY OF HIGH-CURRENT, HIGH-VOLTAGE DRIVERS),			
62			THE CHSTB TABLE COULD PROVIDE ENCODED OUTPUTS, NINE DIGITS, FOR			
63			EXAMPLE, COULD BE ENABLED WITH SIX BITS OF (BUFFERED) OUTPUT:			
64			(001001, 001010, 001100, 010001, 010010, 010100, 100001, 100010, 100100)			
65			IF I/O LINES NEED TO BE CONSERVED, OR IF MANY DIGITS			
66			MUST BE DISPLAYED, AN EXTERNAL DECODER COULD BE ADDED TO THE SYSTEM.			
67			DURING CHARACTER TRANSITIONS A 'BLANK' CHARACTER IS			
68			EXPLICITLY WRITTEN TO THE DISPLAY. THUS,			
69			THERE WILL BE NO CHARACTER 'SHADOWING' CAUSED BY THE			
70			FACT THAT THE HARDWARE OR SOFTWARE DECODER KEEPS ONE			
71			OUTPUT, AND THUS ONE CHARACTER, ACTIVE AT ALL TIMES.			
72						
73			PSGMNT-EIGHT BIT PORT TO ENABLE THE SEVEN SEGMENTS & D-P OF A STANDARD			
74			DISPLAY.			
75			BIT7-BIT0 CORRESPOND TO THE DP AND SEGMENTS G THROUGH A, RESPECTIVELY.			
76			IT IS POSSIBLE TO ACCOMMODATE			
77			DRIVERS WHICH ARE EITHER LOGICALLY INVERTING OR NON-INVERTING BY			
78			SETTING VARIABLE 'SEGPOL' (SEGMENT POLARITY).			
79			NOTE THAT BY HAVING ARBITRARY CONTROL OVER EACH SEGMENT, NON-NUMERIC			
80			CHARACTERS CAN BE REPRESENTED ON A SEVEN SEGMENT DISPLAY.			
81			AS SHOWN IN EXAMPLE SUBROUTINE 'TEST2'.			
82						
83			\$EJECT			

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
		84	: PINPUT-FOUR HIGH-ORDER BITS USED AS INPUTS FROM THE KEYBOARD RETURN LINES
		85	: ASSUMES THAT A KEY DOWN IN THE CURRENTLY ENABLED ROW WOULD RETURN
		86	: A LOW LEVEL
		87	: IN THIS CASE, BIT7 RETURNS THE LEFTMOST COLUMN, BIT4 THE RIGHTMOST.
		88	: THE HIGH-ORDER BITS ARE USED SO THAT IF AN OFF-CHIP DECODER IS USED
		89	: TO ENABLE UP TO 16 CHARACTERS, FOR EXAMPLE, IT COULD BE DRIVEN BY
		90	: THE LOW ORDER BITS OF THE SAME PORT.
		91	: NOTE ALSO THAT IF A SIXTEEN KEY MATRIX WERE ELECTRICALLY ORGANIZED
		92	: IN A 2X8 ARRAY, ONLY TWO RETURN LINES WOULD BE NEEDED.
		93	: (IN THIS CASE, PERHAPS T0 AND T1 COULD BE USED FOR INPUT BITS.)
		94	: PULL-UP RESISTORS ON THE RETURN LINES MIGHT BE IN ORDER IF THERE IS ANY
		95	: POSSIBILITY OF A HIGH-IMPEDENCE CONDUCTIVE PATH THROUGH THE SWITCH WHEN
		96	: IT IS SUPPOSED TO BE 'OPEN'.
		97	: (THIS PHENOMENON HAS ACTUALLY BEEN OBSERVED.)
		98	: THE DRIVERS USED IN THE PROTOTYPE WERE ALL NON-INVERTING IN THAT
		99	: A HIGH LEVEL ON AN OUTPUT LINE IS USED TO TURN A CHARACTER OR SEGMENT ON
		100	: THERE ARE A TOTAL OF SEVEN I/O LINES LEFT OVER
		101	: THE ALGORITHM FOR DRIVING THE DISPLAY USES A BLOCK OF INTERNAL RAM
		102	: AS DISPLAY REGISTERS, WITH ONE BYTE CORRESPONDING TO EACH CHARACTER OF THE
		103	: DISPLAY. THE EIGHT BITS OF EACH BYTE CORRESPOND TO THE SEVEN SEGMENTS & DP
		104	: OF EACH CHARACTER. IF AN EXTERNAL ENCODER IS USED (SUCH AS A FOUR-BIT TO
		105	: SEVEN-SEGMENT ENCODER OR A ROM FOR TRANSLATING ASCII TO
		106	: SIXTEEN-SEGMENT "STARBURST" DISPLAY PATTERNS), THE TABLE ENTRIES WOULD HOLD
		107	: THE CHARACTER CODES. (IN THE FORMER CASE, AN UNUSED BIT COULD BE USED TO
		108	: ENABLE THE D.P.)
		109	: THUS, WRITING CHARACTERS TO THE DISPLAY FROM THE BACKGROUND PROGRAM
		110	: REALLY ENTAILS WRITING THE APPROPRIATE SEGMENT
		111	: PATTERNS TO A DISPLAY REGISTER- THE ACTUAL OUTPUTTING IS AUTOMATIC.
		112	: THE LEFTMOST CHARACTER CORRESPONDS TO THE LAST BYTE OF THE DISPLAY
		113	: REGISTERS, AND IS ACCESSED BY NEXTPL=8 (SEE SOURCE); THE RIGHTMOST
		114	: CHARACTER IS THE FIRST DISPLAY BYTE, WHEN NEXTPL=1.
		115	: UTILITY SUBROUTINES ARE INCLUDED HERE TO TRANSLATE FOUR BIT NUMBERS TO HEX
		116	: DIGIT PATTERNS, AND WRITE THEM INTO THE DISPLAY REGISTERS SEQUENTIALLY
		117	: (EITHER FILLING FROM THE LEFT- H.P. CALCULATOR STYLE OR FROM THE
		118	: RIGHT- T.I. STYLE, SUBROUTINES WDISP AND RENTRY, RESPECTIVELY)
		119	: THE KEYBOARD SCANNING ALGORITHM SHOWN HERE REQUIRES A KEY BE DOWN FOR
		120	: SOME NUMBER OF COMPLETE DISPLAY SCANS TO BE ACKNOWLEDGED. SINCE IT IS
		121	: INTENDED FOR 'ONE-FINGER' OPERATION, TWO-KEY ROLLOVER/N-KEY LOCKOUT HAS
		122	: BEEN IMPLEMENTED. HOWEVER, MODIFICATIONS WOULD BE POSSIBLE TO ALLOW, FOR
		123	: EXAMPLE, ONE KEY IN THE MATRIX TO BE USED AS A SHIFT KEY OR CONTROL KEY
		124	: TO BE HELD DOWN WHILE ANOTHER KEY IN THE MATRIX IS PRESSED (SEE NOTE WITHIN
		125	: THE BODY OF THE LISTING.)
		126	:
		127	:
		128	:
		129	:
		130	:
		131	: \$EJECT

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0000

PAGE 54

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
132			:KBE AWARE THAT NO MORE THAN TWO KEYS CAN EVER BE DOWN UNLESS DIODES			
133			:ARE PLACED IN SERIES WITH ALL OF THE SWITCHES- CERTAINLY NOT THE CASE FOR EL			
134			:CHEAPO KEYBOARDS- BECAUSE SOME COMBINATIONS OF THREE KEYS DOWN WILL RESULT			
135			:IN A 'PHANTOM' FOURTH KEY BEING PERCEIVED			
136			:THE 'PHANTOM' KEY WOULD BE THE FOURTH 'CORNER' WHEN THREE KEYS FORMING			
137			:A RECTANGULAR PATTERN (IN THE X-Y KEY MATRIX) ARE DOWN.)			
138			:IF DIODES ARE PLACED IN THE SCANNING ARRAY, CONSIDERATIONS MUST BE MADE			
139			:ABOUT HOW THE DIODE VOLTAGE DROP WILL AFFECT INPUT LOGIC LEVELS.			
140			:			
141			:WHEN A DEBOUNCED KEY IS DETECTED, THE NUMBER OF ITS POSITION IN THE KEY			
142			:MATRIX (LEFT-TO-RIGHT, BOTTOM-TO-TOP, STARTING FROM 00) IS PLACED INTO			
143			:RAM LOCATION 'KBDBUF'. AN INPUT SUBROUTINE THEN NEED ONLY READ THIS LOCATION			
144			:REPEATEDLY TO DETERMINE WHEN A KEY HAS BEEN PRESSED WHEN A KEY IS DETECTED.			
145			:A SPECIAL CODE BYTE SHOULD BE WRITTEN BACK TO INTO 'KBDBUF' TO PREVENT			
146			:REPEATED DETECTIONS OF THE SAME KEY			
147			:THE ROUTINE 'KBDIN' DEMONSTRATES A TYPICAL INPUT PROTOCOL, ALONG WITH A METHOD			
148			:FOR TRANSLATING A KEY POSITION TO ITS ASSOCIATED SIGNIFICANCE BY ACCESSING			
149			:TABLE 'LEGND' IN ROM			
150			:			
151			:\$EJECT			

LOC	OBJ	SEQ	SOURCE STATEMENT	THRESHOLD	330002	032	100	001
152			*****					
153			INITIAL EQUATES TO DEFINE SYSTEM CONFIGURATION					
154								
155								
156								
157								
0010		158	PDIGIT EQU BUS					
0008		159	PSGMNT EQU F1					
0009		160	PINPUT EQU P2					
		161						
		162						
		163						
		164						
		165						
0000		166	POSLOG EQU 00H					
00FF		167	NEGLOG EQU 0FFH					
		168						
0000		169	CHRPOL EQU POSLOG					
0000		170	SEGPOL EQU POSLOG					
00F0		171	INFMSK EQU 0F0H					
		172						
0008		173	CHARNO EQU 8					
0004		174	NROWS EQU 4					
0004		175	NCOLS EQU 4					
		176						
FFF0		177	TICK EQU -10H					
0004		178	DEBNCE EQU 4					
0000		179	BLANK EQU 00H					
		180						
		181						
		182						
000F		183	ENCMSK EQU 0FH					
		184						
		185	#EJECT					

1515-II MCS-48/UPI-41 MACRO ASSEMBLER: V2.0

PAGE 6

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
		186	*****
		187	
		188	BANK 0 REGISTERS USED
		189	
		190	POINTERS USED FOR INDIRECT RAM ACCESSING:
0000		191	PNTR0 EQU R0
0001		192	PNTR1 EQU R1
0007		193	NEXTPL EQU R7 ;USED TO KEEP TRACK OF CHARACTER POSITION BEING
		194	;WRITTEN INTO
		195	
		196	*****
		197	
		198	BANK 1 REGISTER ALLOCATION
		199	
		200	PNTR0 EQU R0 (ALREADY DEFINED)
		201	PNTR1 EQU R1
0002		202	ASAVE EQU R2 ;HOLDS ACCUMULATOR VALUE DURING SERVICE ROUTINE
0004		203	ROTPAT EQU R4 ;USED TO HOLD INPUT PATTERN BEING ROTATED THROUGH CY
0005		204	ROTCNT EQU R5 ;COUNTS NUMBER OF BITS ROTATED THROUGH CY
0006		205	LASTKY EQU R6 ;HOLDS KEY POSITION OF LAST KEY DEPRESSION DETECTED
0007		206	CURDIG EQU R7 ;HOLDS POSITION OF NEXT CHARACTER TO BE DISPLAYED
		207	
		208	*****
		209	
		210	DATA RAM ALLOCATION
		211	
0020		212	NREPTS EQU 32 ;KEEPS TRACK OF SUCCESSIVE READS OF SAME KEYSTROKE
0021		213	KEYLOC EQU 33 ;INCREMENTED AS SUCCESSIVE KEY LOCATIONS SCANNED
0022		214	KDBUF EQU 34 ;CARRIES POSITION OF DEBOUNCED KEY FROM REFRSH ROUTINE
		215	; \ BACK TO BACKGROUND PROGRAM
0023		216	RDELAY EQU 35 ;NON-ZERO WHEN DISPLAY IN PROGRESS
		217	
		218	THE LAST <CHARNO> REGISTERS HOLD THE DISPLAY SEGMENT PATTERNS
		219	
0037		220	SEGMAP EQU (63-CHARNO) ;BASE OF REGISTER ARRAY FOR DISPLAY PATTERNS
		221	; \ (COULD BE ANYWHERE IN INTERNAL RAM)
		222	
		223	*****
		224	
		225	NOTE THAT LASTKY, CURDIG, AND F1 RETAIN STATUS INFORMATION FROM
		226	ONE INTERRUPT TO THE NEXT. ALL OTHER REGISTERS MAY BE USED IN
		227	THE USER'S OWN INTERRUPT SERVICING ROUTINE
		228	
		229	*****
		230	
		231	\$EJECT

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

PAGE 7

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
		232	*****
		233	*****
		234	*****
0000		235	ORG 000H
0000 0460		236	JMP INIT
		237	*****
		238	*****
		239	*****
		240	*****
0007		241	ORG 007H
		242	*****
		243	TIINT TIMER INTERRUPT SUBROUTINE
		244	CALL MADE TO LOC 007H WHEN TIMER TIMES OUT
		245	TIMER CAN BE RE-INITIALIZED AT THIS POINT IF DESIRED.
		246	USED HERE TO CAUSE THE DISPLAY REFRESH AND KEY SCAN ROUTINES TO
		247	BE CALLED PERIODICALLY.
0007 05		248	TIINT: SEL R61
0008 0A		249	MOV ASAVE, A
0009 23F0		250	MOV R, #TICK
000B 62		251	MOV T, A ;RELOAD TIMER INTERVAL
		252	*****
		253	*****
		254	*****
		255	THE USER'S OWN TIMER INTERRUPT ROUTINE (IF IT EXISTS) COULD
		256	BE PLACED AT THIS POINT
		257	*****
		258	*****
		259	*****
000C 1410		260	CALL REFRSH ;CAUSE DISPLAY TO BE UPDATED
		261	*****
		262	THE COMPLETE INTERRUPT ROUTINE SHOULD BE COPIED HERE
		263	TO SAVE A FULL LEVEL OF SUBROUTINE NESTING.
		264	IT WAS WRITTEN AS A SUBROUTINE HERE FOR THE SAKE OF CLARITY.
		265	*****
		266	*****
		267	*****
000E FA		268	TIRET TIMER INTERRUPT RETURN CODE- RESTORES ACC VALUE
000F 93		269	TIRET: MOV R, ASAVE
		270	RETR
		271	*****
		272	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		273	*****
		274	REFRSH SUBROUTINE TO MULTIPLEX SEVEN-SEGMENT DISPLAYS
		275	EACH CALL CAUSES THE NEXT CHARACTER TO BE DISPLAYED.
		276	ACCORDING TO THE CONTENTS OF THE SEGMAP REGISTER ARRAY
		277	REFRSH SHOULD BE CALLED AT LEAST EVERY MSEC OR SO.
		278	*****
0010 2300		280	REFRSH: MOV A, #BLANK XOR SEGPOL
0012 39		281	OUTL PSGMT, A ; WRITE BLANK PATTERN TO SEG DRIVERS
0013 2357		282	REFR1: MOV A, #CHRSTB ; LOOK UP DIGIT ENABLE PATTERN
0015 6F		283	ADD A, CURDIG ; ADD CURDIG DISPLACEMENT
0016 A3		284	MOV A, @A ; ENABLE ONE BIT OF ACCUMULATOR
0017 02		285	OUTL PDIGIT, A ; ENERGIZE CHARACTER
		286	;
		287	;
		288	MOV A, #SEGMAP ; LOAD BASE OF REGISTER ARRAY
001A 6F		289	ADD A, CURDIG ; ADD CURDIG DISPLACEMENT
001B A9		290	MOV PNTR1, A
001C F1		291	MOV A, @PNTR1 ; LOAD ACC W/ NEXT SEGMENT PATTERN
001D 39		292	OUTL PSGMT, A ; ENABLE APPROPRIATE SEGMENTS
		293	;
		294	*****
		295	THE NEXT CHARACTER IS NOW BEING DISPLAYED.
		296	THE KEYBOARD SCAN ROUTINE IS INTEGRATED INTO THE DISPLAY SCAN.
		297	WITH THE CURRENT ROW ENERGIZED, CHECK IF THERE ARE ANY INPUTS
		298	*****
		299	;
001E B821		300	SCAN: MOV PNTR0, #KEYLOC ; SET POINTER FOR SEVERAL KEYLOC REFERENCES
0020 0A		301	IN A, PINPUT ; LOAD ANY SWITCH CLOSURES
		302	;
		303	*****
		304	## THIS BLOCK OF CODE IS NOT NEEDED BY THE KEYBOARD SCAN LOGIC. ##
		305	## HOWEVER, ITS INCLUSION WOULD SPEED THINGS UP A BIT BY ##
		306	## SKIPPING OVER ROWS IN WHICH NO KEYS ARE DOWN. ##
		307	## IT WAS OMITTED HERE TO CONSERVE ROM SPACE, BUT MIGHT BE ##
		308	## RESTORED IF VERY LARGE KEYBOARDS (ESPECIALLY THOSE WITH EIGHT) ##
		309	## KEYS PER ROW) ARE TO BE USED WITH THIS ALGORITHM. ##
		310	*****
		311	## CPL A ; ANY CLOSURES DETECTED ARE NOW ONE BITS ##
		312	## ANL A, #INPMASK ##
		313	## JNZ SCAN1 ; -IF A KEY IN THE CURRENTLY ENABLED ROW IS DOWN ##
		314	## NO KEY IS NOW DOWN SO THE KEYLOC COUNT MAY BE UPDATED DIRECTLY ##
		315	## MOV A, @PNTR0 ##
		316	## ADD A, #NCOLS ##
		317	## MOV @PNTR0, A ##
		318	## JMP SCAN6 ##
		319	*****
		320	## IF THIS CODE IS USED, SUBSTITUTE THE 'JC SCAN5' FOUR LINES ##
		321	## HENCE WITH 'JNC SCAN5' TO ACCOMMODATE THE INVERTED POLARITY ##
		322	*****
		323	\$EJECT

IS15-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

PAGE 9

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
		324	*****
		325	ROTATE BITS THROUGH THE CY WHILE INCREMENTING KEYLOC.
		326	*****
0021	BD04	327	MOV R0, #0 ; SET UP FOR (NCOLS) LOOPS THROUGH 'NXTLOC'
0023	F7	328	SCAN1: MOV R1, #0 ; SET UP FOR (NCOLS) LOOPS THROUGH 'NXTLOC'
0024	AC	329	NXTLOC: RLC A ; ROTATE CARRY INTO BIT 0
0025	F63F	330	MOV R0, #0 ; SAVE SHIFTED BIT PATTERN
		331	JC SCAN5 ; ONE BIT IN CY INDICATES KEY NOT DOWN
		332	*****
		333	*****
		334	*****
		335	AT THIS POINT IT HAS JUST BEEN DETERMINED THAT THE VALUE
		336	OF KEYLOC IS THE POSITION OF A KEY WHICH IS NOW DOWN
		337	THE FOLLOWING CODE DEBOUNCES THE KEY, ETC.
		338	IF MODIFICATIONS TO THE KEYBOARD LOGIC, I.E. THE INCLUSION
		339	OF A SHIFT, CONTROL, OR MODE KEY IN THE KEY MATRIX ITSELF
		340	ARE DESIRED, THEY SHOULD BE MADE AT THIS POINT, BEFORE
		341	THE DEBOUNCE LOGIC BEGINS. FOR EXAMPLE, AT THIS POINT
		342	KEYLOC COULD BE COMPARED AGAINST THE POSITION OF THE MODE
		343	KEY, AND IF THEY MATCH SET SOME FLAG BIT AND JUMP TO
		344	LABEL 'SCAN5'. OR, BY COMPARING KEYLOC AGAINST THE LAST
		345	KEY DEBOUNCED, IMMEDIATE TWO-KEY ROLLOVER COULD BE
		346	IMPLEMENTED.
		347	*****
		348	*****
		349	*****
0027	A5	350	CLR F1 ; MARK THAT AT LEAST ONE KEY WAS DETECTED
0028	B5	351	CPL F1 ; IN THE CURRENT SCAN
		352	*****
		353	*****
		354	A KEYSTROKE WAS DETECTED FOR THE CURRENT COLUMN. ITS
		355	POSITION IS IN REGISTER KEYLOC. SEE IF SAME KEY SENSED LAST CYCLE.
		356	*****
		357	*****
0029	F0	358	MOV A, @PNTR0 ; PNTR0 STILL HOLDS #KEYLOC
002A	2E	359	XCH A, LASTKY ; EXCHANGE LASTKEY WITH A
002B	DE	360	XRL A, LASTKY ; XOR LASTKEY WITH A
002C	B820	361	MOV PNTR0, #NREPTS ; PREPARE TO CHECK AND/OR MODIFY REPEAT COUNT
002E	C634	362	JZ SCAN5 ; IF ZERO, JUMP TO SCAN5
		363	*****
		364	\$EJECT

IS15-I1 MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

PAGE 10

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
		365	*****
		366	A DIFFERENT KEY WAS READ ON THIS CYCLE THAN ON THE PREVIOUS CYCLE.
		367	SET NREPTS TO THE DEBOUNCE PARAMETER FOR A NEW COUNTDOWN.
		368	*****
		369	
0030	B004	370	MOV @PNTR0, #DEBNCE
0032	043F	371	JMP SCANS
		372	
		373	*****
		374	SAME KEY WAS DETECTED AS ON PREVIOUS CYCLE
		375	LOOK AT NREPTS. IF ALREADY ZERO, DO NOTHING
		376	ELSE DECREMENT NREPTS
		377	IF THIS RESULTS IN ZERO, MOVE LASTKY INTO KBDBUF
		378	*****
		379	
0034	F0	380	SCANS: MOV A, @PNTR0
0035	C63F	381	JZ SCANS ; IF ALREADY ZERO
0037	07	382	DEC A ; INDICATE ONE MORE SUCCESSIVE KEY DETECTION
0038	A0	383	MOV @PNTR0, A
0039	963F	384	JNZ SCANS ; IF DECREMENT DOES NOT RESULT IN ZERO
003B	FE	385	MOV A, LASTKY
003C	B022	386	MOV PNTR0, #KBDBUF
003E	A0	387	MOV @PNTR0, A ; TO MARK NEW KEY CLOSURE
		388	
003F	B821	389	SCANS: MOV PNTR0, #KEYLOC
0041	10	390	INC @PNTR0
0042	FC	391	MOV A, ROTPAT
0043	ED23	392	DJNZ ROTCNT, NXTLOC
		393	
		394	
0045	EF57	395	SCANS: DJNZ CURDIG, SCANS
		396	
		397	\$EJECT

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

PAGE 11

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SER	SOURCE STATEMENT
		398	;
		399	*****
		400	THE FOLLOWING CODE SEGMENT IS USED BY THE KEYBOARD SCANNING ROUTINE
		401	IT IS EXECUTED ONLY AFTER A REFRESH SEQUENCE OF ALL
		402	THE CHARACTERS IN THE DISPLAY IS COMPLETED
		403	*****
		404	;
0047	B08	405	MOV CURDIG, #CHARNO
0049	B000	406	MOV @PNTR0, #0 ;PNTR0 STILL CONTAINS #KEYLOC
004B	764F	407	JF1 SCAN8 ;JUMP IF ANY KEYS WERE DETECTED
004D	BEFF	408	MOV LASTKY, #0FFH ;CHANGE <LASTKY> WHEN NO KEYS ARE DOWN
004F	A5	409	SCAN8: CLR F1
		410	;
		411	*****
		412	THE NEXT CODE SEGMENT IS THE INTERRUPT-DRIVEN PORTION OF THE 'DELAY'
		413	UTILITY: IT DECREMENTS RAM LOCATION 'RDELAY' ONCE PER DISPLAY SCAN
		414	IF 'RDELAY' IS NOT ALREADY ZERO
		415	*****
0050	B923	417	MOV PNTR1, #RDELAY
0052	F1	418	MOV A, @PNTR1
0053	D657	419	JZ SCAN9
0055	07	420	DEC A
0056	A1	421	MOV @PNTR1, A
		422	;
0057	83	423	SCAN9: RET
		424	;
		425	*****
		426	;
		427	CHARSTB IS THE BASE FOR THE PATTERNS TO ENABLE ONE-OF-CHARNO CHARACTERS.
0057		428	CHARSTB EQU (#-1) AND 0FFH
0058	01	429	DB (00000001B XOR CHARPOL)
0059	02	430	DB (00000010B XOR CHARPOL)
005A	04	431	DB (00000100B XOR CHARPOL)
005B	08	432	DB (00001000B XOR CHARPOL)
005C	10	433	DB (00010000B XOR CHARPOL)
005D	20	434	DB (00100000B XOR CHARPOL)
005E	40	435	DB (01000000B XOR CHARPOL)
005F	80	436	DB (10000000B XOR CHARPOL)
		437	;
		438	\$EJECT

ISIS-II MCS-48/UP1-41 MACRO ASSEMBLER, V2.0

PAGE 12

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT	THEM STATEMENT	LOC	OBJ
		439	INIT	INITIALIZES PROCESSOR REGISTERS		
0060	D5	440	INIT	SEL R01		
0061	B009	441	MOV	CURDIG, #CHARNO		
0062	B022	442	MOV	PNTRO, #KBDBUF		
0065	B0FF	443	MOV	@PNTRO, #OFFH		
0067	B021	444	MOV	PNTRO, #KEYLOC		
0069	B000	445	MOV	@PNTRO, #0		
006B	23F0	446	MOV	A, #INPMASK		
006D	3A	447	OUTL	PINPUT, A ; SET BIDIRECTIONAL INPUT LINES		
006E	C5	448	SEL	R00		
006F	149E	449	CALL	CLEAR ; UTILITY FOR SETTING INITIAL DISPLAY REGISTERS.		
0071	A5	450	CLR	F1		
0072	23F0	451	MOV	A, #TICK ; LOAD INTERRUPT RATE VALUE		
0074	62	452	MOV	T, A		
0075	55	453	STRT	T		
0076	25	454	EN	TENTI ; ENABLE TIMER INTERRUPTS		
		455				
		456				
		457	*****			
		458				
		459	ECHO	CHECK FOR ANY NEW KEYSTROKES DETECTED		
		460		TRANSLATE EACH KEYSTROKE INTO A SEGMENT PATTERN		
		461		AND WRITE IT INTO THE APPROPRIATE DISPLAY REGISTER.		
		462				
		463	*****			
		464				
0077	1483	465	ECHO	CALL KBDIN ; GET NEXT KEYSTROKE		
0079	B281	466	JBS	FKEY ; JUMP IF KEY IN RIGHTHAND COLUMN		
		467		SINCE THE ACC IS USED BY ENRACC AND RENTRY, ITS CONTENTS MUST		
		468		BE PROCESSED OR SAVED BEFORE ENRACC IS CALLED		
007B	14BA	469	CALL	ENRACC ; FORM APPROPRIATE SEGMENT PATTERN		
007D	14DB	470	CALL	RENTRY ; WRITE PATTERN INTO DISPLAY REGISTERS		
007F	0477	471	JMP	ECHO ; LOOP INDEFINITELY		
		472				
0081	2400	473	FKEY	JMP FUNCTN ; JUMP TO OFF-PAGE CODE TO CALL DEMO ROUTINE		
		474				
		475	#EJECT			

```

LOC OBJ          SEQ          SOURCE STATEMENT
476 . *****
477 .
478 .      THE FOLLOWING SUBROUTINES IMPLEMENT THE UTILITIES COMMONLY USED FOR
479 .      MOST KEYBOARD/DISPLAY APPLICATIONS.
480 .      THEY COULD BE USED EXACTLY AS SHOWN HERE OR ADAPTED FOR SPECIAL CASES.
481 .
482 . *****
483 .
484 . KBDIN  KEYBOARD INPUT SUBROUTINE.
485 .      COULD BE USED TO INTERFACE THE USER'S BACKGROUND PROGRAM WITH
486 .      THE INTERRUPT DRIVEN KEYBOARD SCANNER.
487 .      RETURNS ONLY AFTER A NEW KEYSTROKE HAS BEEN DETECTED AND DEBOUNCED.
488 .      ENCODED VALUE OF KEY (RATHER THAN ITS POSITION IN SWITCH MATRIX) IS
489 .      RETURNED IN THE ACCUMULATOR.
0083 B922 490 KBDIN: MOV     A, #KBDUF
0085 2380 491     MOV     A, #00H      ; KBDUF WILL BE MARKED AS CLEAR
0087 21      492     XCH     A, @PNTR1    ; LOAD BUFFER VALUE
0088 F283 493     JEB     KBDIN
008A 038E 494     ADD     A, #LEGND5      ; ADD BASE OF KEY ENCODING TABLE
008C A3      495     MOVP    A, @A          ; OBTAIN BYTE REPRESENTING KEY SIGNIFICANCE
008D 83      496     RET
497 .
498 .
499 . LEGND5 IS THE BASE FOR TABLE SHOWING KEY MATRIX SIGNIFICANCE
500 .      FOR THE KEYBOARD USED IN THE PROTOTYPE.
501 .      KEY LAYOUT IS AS SHOWN TO THE RIGHT.
502 .
503 .      NOTE THAT BIT6-BIT4 MAY BE USED TO ENCODE KEY TYPE.  IN THIS CASE:
504 .      BIT4 INDICATES REGULAR DECIMAL DIGITS,
505 .      BIT5 INDICATES RIGHT-COLUMN FUNCTION KEYS,
506 .      BIT6 INDICATES PUNCTUATION MARKS ( * AND # ).
507 .
008E 508 LEGND5 EQU    ($ AND 0FFH) ; USE LOW ORDER BITS AS TABLE INDEX
008E 4F      509     DB      4FH
008F 10      510     DB      10H
0090 4E      511     DB      4EH
0091 28      512     DB      28H      ; PDIGIT4==> 1      2      3      <1>
0092 17      513     DB      17H
0093 18      514     DB      18H      ; PDIGIT5==> 4      5      6      <2>
0094 19      515     DB      19H
0095 24      516     DB      24H      ; PDIGIT6==> 7      8      9      <3>
0096 14      517     DB      14H
0097 15      518     DB      15H      ; PDIGIT7==> *      0      #      <4>
0098 16      519     DB      16H
0099 22      520     DB      22H      ;      !      !      !      !
009A 11      521     DB      11H      ;      !      !      !      !
009B 12      522     DB      12H      ;      V      V      V      V
009C 13      523     DB      13H      ;      PINUT7 PINUT6 PINUT5 PINUT4
009D 21      524     DB      21H
525 $EJECT

```


1515-11 MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

PAGE 14

AP40 INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT	THIRGRTS	CONRAD	322	100	30J
		526	*****					
		527						
		528	:CLEAR	WRITES 'BLANK' CHARACTERS INTO ALL DISPLAY REGISTERS.				
		529	:	RETURNS WITH NEXTPL SET TO LEFTMOST CHARACTER POSITION				
		530	:FILL	WRITES SEGMENT PATTERN NOW IN ACC INTO ALL DISPLAY REGISTERS				
009E	2300	531	CLEAR	MOV A, #BLANK XOR SEGPOL				
00A0	B938	532	FILL	MOV PNTRL, #SEGMAP+1				
00A2	BF08	533		MOV NEXTPL, #CHARNO				
00A4	A1	534	CLR1	MOV @PNTRL, A ; STORE THE BLANK CODE				
00A5	19	535	INC	PNTRL, 1 ; POINT TO NEXT CHARACTER TO THE LEFT				
00A6	EFA4	536	DJNZ	NEXTPL, CLR1				
00A8	BF08	537		MOV NEXTPL, #CHARNO				
00AA	83	538		RET				
		539	:					
		540	*****					
		541	:					
		542	:PRINT	SUBROUTINE TO COPY A STRING OF BIT PATTERNS FROM ROM TO THE				
		543	:	DISPLAY REGISTERS. STRING STARTS AT LOCATION POINTED TO BY PNTR0.				
		544	:	CONTINUES UNTIL AN ESCAPE CODE (OFFH) IS REACHED.				
		545	:	NOTE THAT THE CHARACTER STRING PUT OUT MUST BE LOCATED ON THE SAME				
		546	:	PAGE AS THIS SUBROUTINE, SINCE SAME-PAGE MOVES ARE USED.				
		547	:	PRINT IN TURN CALLS EITHER SUBROUTINE 'WDISP' OR 'RETRY'				
		548	:	TO ACTUALLY EFFECT WRITING INTO THE DISPLAY REGISTERS.				
00AB	F8	549	PRINT:	MOV A, PNTR0 ; LOAD NEXT CHARACTER LOCATION				
00AC	A3	550		MOVP A, @A ; LOAD BIT PATTERN INDIRECT				
00AD	C6B4	551		JZ PNTRL, #ESCAPE PATTERN				
00AF	14D0	552		CALL WDISP ; OUTPUT TO NEXT CHARACTER POSITION				
		553	:##	CALL RENTRY ; INSTEAD IF MESSAGE IS TO BE RIGHT JUSTIFIED)				
00B1	18	554		INC PNTR0 ; INDEX POINTER				
00B2	04AB	555		JMP PRINT				
00B4	83	556	PNTRL	PET ; DONE				
		557	:					
		558	*****					
		559	:					
		560	:JOHN	ARRAY HOLDS THE BIT PATTERNS FOR THE LETTERS 'JOHN' (SEE 'TEST2')				
		561	:	(NOTE THAT 'OHN' IS WRITTEN IN LOWER CASE LETTERS)				
00B5		562	JOHN	EQU \$ AND OFFH				
00B5	1E	563		DB 00011110B XOR SEGPOL				
00B6	5C	564		DB 01011100B XOR SEGPOL				
00B7	74	565		DB 01101000B XOR SEGPOL				
00B8	54	566		DB 01010100B XOR SEGPOL				
00B9	00	567		DB 00				
		568	:					
		569	\$EJECT					

LOC	OBJ	SEQ	SOURCE STATEMENT
		570	*****
		571	
		572	ENCACC ENCODES LSNIFFLE OF ACC INTO HEX BIT PATTERN INTO ACC
00BA 530F		573	ENCACC ANL A, #ENCMASK
00BC 03C0		574	ADD A, #DGPATS
00BE A3		575	MOV A, @A
00BF 83		576	RET
		577	DGPATS IS THE BASE FOR THE TABLE OF SEGMENT PATTERNS FOR THE BASIC
		578	DIGITS. HERE THE FULL HEX SET (0-F) IS INCLUDED.
		579	FOR MANY USER APPLICATIONS, THE CHARACTER SET MAY BE AMENDED OR AUGMENTED
		580	TO INCLUDE ADDITIONAL SPECIAL PURPOSE PATTERNS.
		581	FORMAT IS PGFEDCBA IN STANDARD SEVEN-SEGMENT ENCODING CONVENTION
		582	WHERE P REPRESENTS THE DECIMAL POINT.
00C0		583	DGPATS EQU \$ AND OFFH
00C0 3F		584	DB 00111111B XOR SEGOL
00C1 06		585	DB 00000110B XOR SEGOL
00C2 58		586	DB 01011011B XOR SEGOL
00C3 4F		587	DB 01001111B XOR SEGOL
00C4 66		588	DB 01100110B XOR SEGOL
00C5 6D		589	DB 01101101B XOR SEGOL
00C6 7D		590	DB 01111101B XOR SEGOL
00C7 07		591	DB 00000111B XOR SEGOL
00C8 7F		592	DB 01111111B XOR SEGOL
00C9 67		593	DB 01100111B XOR SEGOL
00CA 77		594	DB 01110111B XOR SEGOL
00CB 7C		595	DB 01111100B XOR SEGOL
00CC 39		596	DB 00111001B XOR SEGOL
00CD 5E		597	DB 01011110B XOR SEGOL
00CE 79		598	DB 01111001B XOR SEGOL
00CF 71		599	DB 01110001B XOR SEGOL
		600	
		601	*****
		602	
		603	WDISP WRITES BIT PATTERN NOW IN ACC INTO NEXT CHARACTER POSITION
		604	OF THE DISPLAY (NEXTPL). ADJUSTS NEXTPL POINTER VALUE
		605	RESULTS IN DISPLAY BEING FILLED LEFT TO RIGHT, THEN RESTARTING
00D0 A9		606	WDISP MOV PNTR1, A
00D1 FF		607	MOV A, NEXTPL
00D2 0337		608	ADD A, #SEGMAP
00D4 29		609	XCH A, PNTR1
00D5 A1		610	MOV @PNTR1, A
00D6 EFDA		611	DJNZ NEXTPL, WDISP1
00D8 BF08		612	MOV NEXTPL, #CHARNO
00DA 83		613	WDISP1 RET
		614	
		615	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		616	*****
		617	
		618	:RENTY SUBROUTINE TO ENTER ACC CONTENTS INTO THE RIGHTMOST DIGIT
		619	: AND SHIFT EVERYTHING ELSE ONE PLACE TO THE LEFT
000B	B938	620	:RENTY: MOV PNT1, #SEGMAP+1
000D	BF08	621	: MOV NEXTPL, #CHARNO
000F	21	622	:RENT1: XCH A, @PNT1
00E0	19	623	: INC PNT1
00E1	EFDF	624	: DJNZ NEXTPL, RENT1
00E3	BF08	625	: MOV NEXTPL, #CHARNO :POINT TO LEFTMOST CHARACTER
00E5	83	626	: RET
		627	
		628	*****
		629	
		630	:RDPADD TOGGLE DECIMAL POINT IN LAST CHARACTER DISPLAY CHARACTER
		631	: DPADD TOGGLES DECIMAL POINT IN THE CHARACTER POINTED TO BY THE ACC
		632	
00E6	2301	633	:RDPADD: MOV A, #01H :SET INDEX TO RIGHTMOST POSITION
00E8	0337	634	: DPADD: ADD A, #SEGMAP :ACCESS DISPLAY REGISTER FOR DESIRED PLACE
00EA	A9	635	: MOV PNT1, A
00EB	F1	636	: MOV A, @PNT1
00EC	D380	637	: XRL A, #00H
00EE	A1	638	: MOV @PNT1, A
00EF	83	639	: RET
		640	
		641	*****
		642	
		643	:HOLD SUBROUTINE CALLED WHEN KEY IS KNOWN TO BE DOWN.
		644	: WILL NOT RETURN UNTIL KEY IS RELEASED.
00F0	05	645	:HOLD: SEL RB1
00F1	FE	646	: MOV A, LASTKY :LASTKY=OFFH IFF NO KEYS DOWN
00F2	05	647	: SEL RB0
00F3	37	648	: CPL A
00F4	96F0	649	: JNZ HOLD
00F6	83	650	: RET
		651	
		652	*****
		653	
		654	:DELAY SUBROUTINE HANGS UP FOR THE NUMBER OF COMPLETE DISPLAY SCANS EQUAL
		655	: TO THE CONTENTS OF THE ACCUMULATOR WHEN CALLED.
00F7	B923	656	:DELAY: MOV PNT1, #DELAY
00F9	A1	657	: MOV @PNT1, A
00FA	F1	658	:DELAY1: MOV A, @PNT1
00FB	96FA	659	: JNZ DELAY1
00FD	83	660	: RET
		661	:EJECT

ISIS-II MCS-48/UP1-41 MACRO ASSEMBLER, V2.0

PAGE 17

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

```

LOC 0B1      SED      SOURCE STATEMENT      TARGET STATEMENT      000      100 00J

0100      662 ORG      100H
063 :
064 : *****
065 :
066 : THE CODE ON THIS PAGE IS FOR DEMONSTRATION PURPOSES ONLY-
067 : I TRULY DOUBT WHETHER ANY END USERS WOULD LIKE TO SEE A NAME
068 : POPPING UP ON THEIR CALCULATOR SCREENS.
069 : HOWEVER, THE CODE SHOWN HERE DOES INDICATE HOW THE UTILITY SUBROUTINES
070 : INCLUDED HERE COULD BE ACCESSED.
071 : THE ROUTINES THEMSELVES ARE CALLED WHEN ONE OF THE FOUR BUTTONS
072 : ON THE RIGHT-HAND SIDE OF THE PROTOTYPE KEYBOARD IS PRESSED.
073 :
074 : *****
075 :
076 : FUNCTN ROUTINE TO IMPLEMENT ONE OF FOUR DEMO UTILITIES, ACCORDING
077 : TO WHICH OF THE FOUR FUNCTION KEYS WAS PRESSED
0100 1212      678 FUNCT: J80      FUNCT1
0102 320E      679      J81      FUNCT2
0104 520A      680      J82      FUNCT3
061 :
0106 14E6      682 FUNCT4: CALL  RDPADD
0108 0477      683      JMP      ECHO
064 :
010A 342E      685 FUNCT3: CALL  TEST3
010C 0477      686      JMP      ECHO
067 :
010E 3424      688 FUNCT2: CALL  TEST2
0110 0477      689      JMP      ECHO
069 :
0112 3416      691 FUNCT1: CALL  TEST1
0114 0477      692      JMP      ECHO
063 :
064 : *****
065 :
066 : TEST1 CODE SEGMENT TO FILL DISPLAY REGISTERS WITH DIGITS DOWN TO '1'
0116 BF08      697 TEST1: MOV      NEXTPL, #CHARNO
0118 B808      698      MOV      PNTR0, #CHARNO ; SET FOR EIGHT LOOP REPETITIONS
011A FF        699 TST11: MOV      A, NEXTPL
011B 14BA      700      CALL  ENCRCC
011D 14D0      701      CALL  WDISP
011F E81A      702      DJNZ  PNTR0, TST11 ; COPY NEXT DIGIT INTO DISPLAY REGISTERS
0121 BF08      703      MOV      NEXTPL, #CHARNO
0123 83        704      RET
065 :
0706 $EJECT

```


1515-11 MCS-48/UPI-41 MACRO ASSEMBLER, V2.0 PAGE 18
 AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

```

LOC OBJ      SEQ      SOURCE STATEMENT
707 ; *****
708 ;
709 :TEST2    WRITES THE SEGMENT PATTERN FOR 'JOHN' ONTO THE DISPLAY.
710 :        WAITS FOR A WHILE, AND THEN CLEARS THE DISPLAY
0124 B8B5    711 TEST2:  MOV     PNTR0, #JOHN
0126 14AB    712        CALL    PRINT
0128 2364    713        MOV     A, #100 ; SCAN DISPLAY FOR 100 CYCLES
012A 14F7    714        CALL    DELAY
012C 049E    715        JMP     CLEAR
716 ;
717 ; *****
718 ;
719 :TEST3    SUBROUTINE TO FILL DISPLAY WITH DASHES
720 :        JUMPS INTO SUBROUTINE 'CLEAR'
721 :        AS SOON AS THE KEY IS RELEASED.
012E 2340    722 TEST3:  MOV     A, #01000000B XOR SEGPOL ; PATTERN FOR '-'
0130 14AB    723        CALL    FILL
0132 14F0    724        CALL    HOLD
0134 049E    725        JMP     CLEAR
726 ;
727 ; *****
728 ;
729 END

USER SYMBOLS
ASAVE 0002  BLANK 0000  CHARN0 0008  CHRPOL 0000  CHRSTB 0057  CLEAR 009E  CLR1 00A4  CURDIG 0007
DEBNCE 0004  DELAY 00F7  DELAY1 00FA  DGPATS 00C0  DPADD 00E8  ECHO 0077  ENCACC 00BA  ENCMASK 000F
FILL 00A0  FKEY 0081  FUNCT1 0112  FUNCT2 010E  FUNCT3 010A  FUNCT4 0106  FUNCIN 0100  HOLD 00F0
INIT 0060  INPMASK 00F0  JOHN 00B5  KBDUF 0022  KBDIN 0083  KEYLOC 0021  LASTKY 0006  LEGNDS 008E
NCOLS 0004  NEGLOG 00FF  NEXTPL 0007  NREPTS 0020  NROWS 0004  NXTLOC 0023  PDIGIT 0010  PINPUT 0009
PNTR0 0000  PNTR1 0001  POSLOG 0000  PRINT 00AB  PRNT1 00B4  PSGMTN 0008  RDELAY 0023  RDPADD 00E6
REFR1 0013  REFRSH 0010  RENTR1 00DF  RENTRY 00DB  ROTCNT 0005  ROTPAT 0004  SCAN 001E  SCAN1 0021
SCANS 0034  SCANS 003F  SCAN6 0045  SCAN8 004F  SCANS 0057  SEGMAP 0037  SEGPOL 0000  TEST1 0116
TEST2 0124  TEST3 012E  TICK 00F0  TIINT 0007  TIRET 000E  TST11 011A  WDISP 0000  WDISP1 00DA

```

ASSEMBLY COMPLETE, NO ERRORS

1-48



APPLICATION NOTE

AP-49

Serial I/O and Math Utilities for the 8049 Microcomputer

Lionel Smith and Cecil Moore
Microcomputer Applications

INTRODUCTION

The Intel® MCS-48 family of microcomputers marked the first time an eight bit computer with program storage, data storage, and I/O facilities was available on a single LSI chip. The performance of the initial processors in the family (the 8748 and the 8048) has been shown to meet or exceed the requirements of most current applications of microcomputers. A new member of the family, however, has been recently introduced which promises to allow the use of the single chip microcomputer in many application areas which have previously required a multichip solution. The Intel® 8049 virtually doubles processing power available to the systems designer. Program storage has been increased from 1K bytes to 2K bytes, data storage has been increased from 64 bytes to 128 bytes, and processing speed has been increased by over 80%. (The 2.5 microsecond instruction cycle of the first members of the family has been reduced to 1.36 microseconds.)

It is obvious that this increase in performance is going to result in far more ambitious programs being written for execution in a single chip microcomputer. This article will show how several program modules can be designed using the 8049. These modules were chosen to illustrate the capability of the 8049 in frequently encountered design situations. The modules included are full duplex serial I/O, binary multiply and divide routines, binary to BCD conversions, and BCD to binary conversion. It should be noted that since the 8049 is totally software compatible with the 8748 and 8048 these routines will also be useful directly on these processors. In addition the algorithms for these programs are expressed in a program design language format which should allow them to be easily understood and extended to suit individual applications with minimal problems.

FULL DUPLEX SERIAL COMMUNICATIONS

Serial communications have always been an important facet in the application of microprocessors. Although this has been partially due to the necessity of connecting a terminal to the microprocessor based system for program generation and debug, the main impetus has been the simple fact that a large share of microprocessors find their way into end products (such as intelligent terminals) which themselves depend on serial communication. When it is necessary to add a serial link to a microprocessor such as the Intel® MCS-85 or 86 the solution is easy; the Intel® 8251A USART or 8273 SDLC chip can easily be added to provide the necessary protocol. When it is necessary to do the same thing to a single chip microcomputer, however, the situation becomes more difficult.

Some microcomputers, such as the Intel 8048 and 8049 have a complete bus interface built into them which allows the simple connection of a USART to the processor chip. Most other single chip microcomputers, although lacking such a bus, can be connected to a USART with various artificial hardware and software constructs. The difficulty with using these chips,

however, is more economic than technical; these same peripheral chips which are such a bargain when coupled to a microprocessor such as the MCS-85 or 86, have a significant cost impact on a single chip microcomputer based system. The high speed of the 8049, however, makes it feasible to implement a serial link under software control with no hardware requirements beyond two of the I/O pins already resident on the microcomputer.

There are many techniques for implementing serial I/O under software control. The application note "Application Techniques for the MCS-48 Family" describes several alternatives suitable for half duplex operation. Full duplex operation is more difficult, however, since it requires the receive and transmit processes to operate concurrently. This difficulty is made more severe if it is necessary for some other process to also operate while serial communication is occurring. Scanning a keyboard and display, for example, is a common operation of single chip microcomputer based system which might have to occur concurrently with the serial receive/transmit process. The next section will describe an algorithm which implements full duplex serial communication to occur concurrently with other tasks. The design goal was to allow 2400 baud, full duplex, serial communication while utilizing no more than 50% of the available processing power of the high speed 8049 microcomputer.

The format used for most asynchronous communication is shown in Figure 1. It consists of eight data bits with a leading 'START' bit and one or more trailing 'STOP' bits. The START bit is used to establish synchronization between the receiver and transmitter. The STOP bits ensure that the receiver will be ready to synchronize itself when the next start bit occurs. Two stop bits are normally used for 110 baud communication and one stop bit for higher rates.

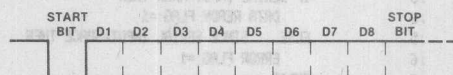


Figure 1.

The algorithm used for reception of the serial data is shown in Figure 2. It uses the on board timer of the 8049 to establish a sampling period of four times the desired baud rates. For 2400 baud operation a crystal frequency of 9.216 MHz was chosen after the following calculation:

$$f = 480N(2400)(4)$$

where 480 is the factor by which the crystal frequency is divided within the processor to get the basic interrupt rate

2400 is the desired baud rate

4 is the required number of samples per bit time

N is the value loaded into the MCS-48 timer when it overflows

The value N was chosen to be two (resulting in $f = 9.216$ MHz) so that the operating frequency of the 8049 could be as high as possible without exceeding the maximum frequency specification of the 8049 (11 MHz).

```

START OF RECEIVE ROUTINE
=====
1 IF RECEIVE FLAG=0 THEN
2   IF SERIAL INPUT=SPACE THEN
3     RECEIVE FLAG:=1
4     BYTE FINISHED FLAG:=0
5   ENDIF
6 ELSE SINCE RECEIVE FLAG=1 THEN
7   IF SYNC FLAG=0 THEN
8     IF SERIAL INPUT=SPACE THEN
9       SYNC FLAG:=1
10      DATA:=80H
11      SAMPLE CNTR:=4
12    ELSE SINCE SERIAL INPUT=MARK THEN
13      RECEIVE FLAG:=0
14    ENDIF
15 ELSE SINCE SYNC FLAG=1 THEN
16   SAMPLE COUNTER:=SAMPLE COUNTER-1
17   IF SAMPLE COUNTER=0 THEN
18     SAMPLE COUNTER:=4
19   IF BYTE FINISHED FLAG=0 THEN
20     CARRY:=SERIAL INPUT
21     SHIFT DATA RIGHT WITH CARRY
22   IF CARRY=1 THEN
23     OKDATA:=DATA
24     IF DATA READY FLAG=0 THEN
25       BYTE FINISHED FLAG:=1
26     ELSE
27       BYTE FINISHED FLAG:=1
28       OVERRUN FLAG:=1
29     ENDIF
30   ENDIF
31 ELSE SINCE BYTE FINISHED FLAG=1 THEN
32   IF SERIAL INPUT=MARK THEN
33     DATA READY FLAG:=1
34   ELSE SINCE SERIAL INPUT=SPACE THEN
35     ERROR FLAG:=1
36   ENDIF
37 RECEIVE FLAG:=0
38 SYNC FLAG:=0
39 ENDIF
40 ENDIF
41 ENDIF

```

Figure 2

The timer interrupt service routine always loads the timer with a constant value. In effect the timer is used to generate an independent time base of four times the required baud rate. This time base is free running and is never modified by either the receive or transmit programs, thus allowing both of them to use the same timer. Routines which do other time dependent tasks (such as scanning keyboards) can also be called periodically at some fixed multiple of this basic time unit.

The algorithm shown in Figure 2 uses this basic clock plus a handful of flags to process the serial input data.

Once the meaning of these flags are understood the operation of the algorithm should be clear. The **Receive Flag** is set whenever the program is in the process of receiving a character. The **Synch Flag** is set when the center of the start bit has been checked and found to be a SPACE (if a MARK is detected at this point the receiver process has been triggered by a noise pulse so the program clears the **Receive Flag** and returns to the idle state). When the program detects synchronization it loads the variable **DATA** with 80H and starts sampling the serial line every four counts. As the data is received it is right shifted into variable **DATA**; after eight bits have been received the initial one set into **DATA** will result in a carry out and the program knows that it has received all eight bits. At this point it will transfer all eight bits to the variable **OKDATA** and set the **Byte Finished Flag** so that on the next sample it will test for a valid stop bit instead of shifting in data. If this test is successful the **Data Ready Flag** will be set to indicate that the data is available to the main process. If the test is unsuccessful the **Error Flag** will be set.

The transmit algorithm is shown in Figure 3. It is executed immediately following the receive process. It is a simple program which divides the free running clock down and transmits a bit every fourth clock. The variable **TICK COUNTER** is used to do the division. The **Transmitting Flag** indicates when a character transmission is in progress and is also used to determine when the START bit should be sent. The **TICK COUNTER** is used to determine when to send the next bit (TICK COUNTER MODULO 4 = 0) and also when the STOP bits should be sent (TICK COUNTER = 9 4). After the transmit routine completes any other timer based routines, such as a keyboard/display scanner or a real time clock, can be executed.

```

START OF TRANSMIT ROUTINE
=====
1 TICK COUNTER:=TICK COUNTER+1
2 IF TICK COUNTER MOD 4=0 THEN
3   IF TRANSMITTING FLAG=1 THEN
4     IF TICK COUNTER=00 1010 00 BINARY THEN
5       TRANSMITTING FLAG:=0
6     ELSE IF TICK COUNTER=00 1001 00 BINARY THEN
7       SEND END MARK
8       TRANSMITTING FLAG:=0
9     ELSE SINCE TICK COUNTER>THE ABOVE COUNT THEN
10      SEND NEXT BIT
11    ENDIF
12 ELSE SINCE TRANSMITTING FLAG=0 THEN
13   IF TRANSMIT REQUEST FLAG=1 THEN
14     XATBYT:=XATBYT
15     TRANSMIT REQUEST FLAG:=0
16     TRANSMITTING FLAG:=1
17     TICK COUNTER:=0
18     SEND SYNC BIT (SPACE)
19   ENDIF
20 ENDIF
21 ENDIF

```

Figure 3

Figure 4 shows the complete receive and transmit programs as they are implemented in the instruction set of

the 8049. Also included in Fig. 4 is a short routine which was used to test the algorithm.

ISIS-II MCS-48/UP1-41 MACRO ASSEMBLER, V2.0

```

LOC OBJ SEQ SOURCE STATEMENT
1 ;*****
2 ;*
3 ;* THIS PROGRAM TESTS THE FULL DUPLEX COMMUNICATION SOFTWARE
4 ;*
5 ;*****
6
7 $INCLUDE(.F1.URTEST.PDL)
8
9 START OF TEST ROUTINE
10 =====
11
12
13
14
15
16 ;1 ERROR COUNT:=0
17 ;1 REPEAT
18 ;2 PATTERN:=0
19 ;2 INITIALIZE TIMER
20 ;2 CLEAR FLAGBYTE
21 ;2 FLAG1=MARK
22 ;2 REPEAT
23 ;3 IF TRANSMIT REQUEST FLAG=0 THEN
24 ;4 NxtBYTE:=PATTERN
25 ;4 TRANSMIT REQUEST FLAG=1
26 ;3 ENDIF
27 ;3 IF DATA READY FLAG=1 THEN
28 ;4 PATTERN:=OKDATA
29 ;4 DATA READY FLAG:=0
30 ;3 ENDIF
31 ;2 UNTIL ERROR FLAG OR OVERRUN FLAG
32 ;2 INCREMENT ERROR COUNT
33 ;1 UNTIL FOREVER
34 EOP
35 $EJECT
0000 36 ORG 0
0000 C5 37 ;1 SELECT REGISTER BANK 0
38 SEL R0
39 ;1 GOTO TEST
0001 2400 40 JMP TEST
41 $ INCLUDE(.F1.UART)
42
43
44 ASYNCHRONOUS RECEIVE/TRANSMIT ROUTINE
45 =====
46 THIS ROUTINE RECEIVES SERIAL CODE USING PIN T0 AS RXD
47 AND CONCURRENTLY TRANSMITS USING PIN P27
48 NOTE
49 THIS ROUTINE USES FLAG 1 TO BUFFER THE TRANSMITTED

```

Figure 4

```

LOC 0B1 SEQ SOURCE STATEMENT
= 50 ; 1 DATA LINE. THIS ELIMINATES THE JITTER THAT
= 51 ; 1 WOULD BE CAUSED BY VARIATIONS IN THE RECEIVE
= 52 ; 1 TIMING. NO OTHER PROGRAM MAY USE FLAG 1 WHILE
= 53 ; 1 THE TIMER INTERRUPT IS ENABLED
= 54 ;
= 55 ;
= 56 ;
= 57 ;
= 58 ;
= 59 ; REGISTER ASSIGNMENTS-BANK1
= 60 ; =====
= 61 ;
= 62 ;
0007 = 63 ATEMP EQU R7 ; USED TO SAVE ACCUMULATOR CONTENTS DURING INTERRUPT
0006 = 64 FLGBYT EQU R6 ; CONTAINS VARIOUS FLAGS USED TO CONTROL THE RECEIVE
= 65 ; AND TRANSMIT PROCESS. SEE CONSTANT DEFINITIONS FOR
= 66 ; THE MEANING OF EACH BIT
0005 = 67 SAMCTR EQU R5 ; SAMPLE COUNTER FOR THE RECEIVE PROCESS
0004 = 68 TCKCTR EQU R4 ; SAMPLE COUNTER FOR THE TRANSMIT PROCESS
0000 = 69 REG0 EQU R0 ; USED AS POINTER REGISTER
= 70 ;
= 71 ; RAM ASSIGNMENTS
= 72 ; =====
= 73 ;
0020 = 74 MOKDAT EQU 20H ; RECEIVE RETURNS VALID DATA IN THIS BYTE
0021 = 75 MDATA EQU 21H ; RECEIVE ACCUMULATES DATA IN THIS BYTE
0022 = 76 MXMTBY EQU 22H ; CONTAINS BYTE BEING TRANSMITTED
0023 = 77 MNXTBY EQU 23H ; CONTAINS THE NEXT BYTE TO BE TRANSMITTED
= 78 $EJECT
= 79 ;
= 80 ;
= 81 ; CONSTANTS
= 82 ; =====
= 83 ;
= 84 ; THE FOLLOWING CONSTANTS ARE USED TO ACCESS THE FLAG BITS CONTAINED
= 85 ; IN REGISTER FLGBYT
= 86 ;
0001 = 87 RCVFLG EQU 01H ; SET WHEN START BIT IS FIRST DETECTED
= 88 ; RESET WHEN RECEIVE PROCESS IS COMPLETE
0002 = 89 SYNFLG EQU 02H ; SET WHEN START BIT IS VERIFIED
= 90 ; RESET WHEN RECEIVE PROCESS IS COMPLETE
0004 = 91 BYFNFL EQU 04H ; RESET WHEN START BIT IS FIRST DETECTED
= 92 ; SET WHEN THE EIGHT DATA BITS HAVE ALL BEEN RECEIVED
0008 = 93 DRDYFL EQU 08H ; SHOULD BE RESET BY MAIN PROGRAM WHEN DATA IS ACCEPTED
= 94 ; SET BY RECEIVE PROCESS WHEN STOP BIT(S) ARE VERIFIED
0010 = 95 ERRFLG EQU 10H ; SHOULD BE RESET BY MAIN PROGRAM WHEN SAMPLED
= 96 ; SET BY RECEIVE PROCESS IF A FRAMING ERROR IS DETECTED
0020 = 97 TRQFL EQU 20H ; TESTED BY MAIN PROGRAM TO DETERMINE IF READY TO
= 98 ; TRANSMIT A NEW BYTE-SET TO INDICATE THAT NXTBYT
= 99 ; HAS BEEN LOADED
= 100 ; RESET BY TRANSMIT PROCESS WHEN BYTE IS ACCEPTED
0040 = 101 TRNGFL EQU 40H ; SET WHEN TRANSMISSION OF A BYTE STARTS
= 102 ; RESET WHEN STOP BIT IS TRANSMITTED
0080 = 103 OVRUN EQU 80H ; SET BY RECEIVE PROCESS WHEN OVERUN OCCURS
= 104 ; SHOULD BE RESET BY MAIN PROGRAM WHEN SAMPLED

```

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
		= 105 ;	
		= 106 ;	GENERAL CONSTANTS
		= 107 ;	=====
		= 108 ;	
0000		= 109 MARK EQU 80H ;	USED TO GENERATE A MARK
FF7F		= 110 SPACE EQU NOT 80H ;	USED TO GENERATE A SPACE
0000		= 111 STPBTS EQU 0 ;	CONTROLS THE NUMBER OF STOP BITS
		= 112	0 GENERATES ONE STOP BIT
		= 113	1 GENERATES TWO STOP BITS
		= 114 ;	
		= 115 #EJECT	
		= 116 ;	
		= 117 ;	START OF RECEIVE/TRANSMIT INTERRUPT SERVICE ROUTINE
		= 118 ;	=====
		= 119 ;	
0007		= 120 ORG 0007H	
		= 121	
		= 122 ;1 ENTER INTERRUPT MODE	
0007 160A		= 123 TISR: JTF UART	
0009 93		= 124 RETR	
000A D5		= 125 UART: SEL RB1	
		= 126 ;1 SAVE ACCUMULATOR CONTENTS	
000B AF		= 127 MOV ATEMP, A	
		= 128 ;1 RELOAD TIMER	
000C 23FE		= 129 MOV A, #TIMCNT	
000E 62		= 130 MOV T, A	
		= 131 ;	
		= 132 ;	OUTPUT TXD BUFFER (F1) TO TXD I/O LINE (P27)
		= 133 ;	=====
		= 134 ;	
000F 7615		= 135 JF1 OMARK	
0011 9A7F		= 136 OSPACE: ANL P2, #SPACE	
0013 0417		= 137 JMP RCV000	
0015 8A00		= 138 OMARK: ORL P2, #MARK	
		= 139 ;	
		= 140 ;	START OF RECEIVE ROUTINE
		= 141 ;	=====
		= 142 ;	
		= 143 ;1 IF RECEIVE FLAG=0 THEN	
0017 FE		= 144 RCV000: MOV A, FLGBYT	
0018 1224		= 145 JB0 RCV010	
		= 146 ;2 IF SERIAL INPUT=SPACE THEN	
001A 3664		= 147 JB0 XMIT	
		= 148 ;3 RECEIVE FLAG:=1	
001C FE		= 149 MOV A, FLGBYT	
001D 4301		= 150 ORL A, #RCVFLG	
		= 151 ;3 BYTE FINISHED FLAG:=0	
001F 53FB		= 152 ANL A, #NOT BYFNFL	
		= 153 ;2 ENDIF	
0021 AE		= 154 MOV FLGBYT, A	
0022 0464		= 155 JMP XMIT	
		= 156 ;1 ELSE SINCE RECEIVE FLAG=1 THEN	
		= 157 ;2 IF SYNC FLAG=0 THEN	
0024 3238		= 158 RCV010: JB1 RCV030	
		= 159 ;3 IF SERIAL INPUT=SPACE THEN	

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	300002	032	100	00J
0026	3633	= 160	JT0 RCV020	001 =			
		= 161 ; 4	SYNC FLAG =1	001 =			
0028	4302	= 162	ORL A,#SYNFLAG	001 =			
002A	AE	= 163	MOV FLGBYT,A	001 =			
		= 164 ; 4	DATA:=80H	001 =			0000
002B	B821	= 165	MOV R0,#MDATA	001 =			0000
002D	B080	= 166	MOV @R0,#80H	001 =			0000
		= 167 ; 4	SAMPLE CNTR:=4	001 =			
002F	B084	= 168	MOV SAMCTR,#4	001 =			
0031	0464	= 169	JMP XMIT	001 =			
		= 170 ; 3	ELSE SINCE SERIAL INPUT=MARK THEN	001 =			
		= 171 ; 4	RECEIVE FLAG:=0	001 =			
0033	53FE	= 172	RCV020: ANL A,#NOT RCVFLAG	001 =			
		= 173 ; 3	ENDIF	001 =			
0035	AE	= 174	MOV FLGBYT,A	001 =			
0036	0464	= 175	JMP XMIT	001 =			0000
		= 176 ; 2	ELSE SINCE SYNC FLAG=1 THEN	001 =			
		= 177 ; 3	SAMPLE COUNTER:=SAMPLE COUNTER-1	001 =			
0038	ED64	= 178	RCV030: DJNZ SAMCTR,XMIT	001 =			0000
		= 179 ; 3	IF SAMPLE COUNTER=0 THEN	001 =			0000
		= 180 ; 4	SAMPLE COUNTER:=4	001 =			0000
003A	B084	= 181	MOV SAMCTR,#4	001 =			
		= 182 ; 4	IF BYTE FINISHED FLAG=0 THEN	001 =			
003C	5259	= 183	JB2 RCV050	001 =			
003E	97	= 184	CLR C	001 =			0000
		= 185 ; 5	CARRY:=SERIAL INPUT	001 =			0000
003F	2642	= 186	JNT0 RCV040	001 =			
0041	A7	= 187	CPL C	001 =			
0042	B821	= 188	RCV040: MOV R0,#MDATA	001 =			
0044	F0	= 189	MOV A,@R0	001 =			
		= 190 ; 5	SHIFT DATA RIGHT WITH CARRY	001 =			0000
0045	67	= 191	RRC A	001 =			0000
0046	A0	= 192	MOV @R0,A	001 =			0000
		= 193 ; 5	IF CARRY=1 THEN	001 =			0000
0047	E664	= 194	JNC XMIT	001 =			
		= 195 ; 6	OKDATA:=DATA	001 =			
0049	B820	= 196	MOV R0,#OKDAT	001 =			
004B	A0	= 197	MOV @R0,A	001 =			
		= 198 ; 6	IF DATA READY FLAG=0 THEN	001 =			
004C	FE	= 199	MOV TVR,A:FLGBYT	001 =			0000
004D	7254	= 200	JB3 RCV045	001 =			0000
		= 201 ; 7	BYTE FINISHED FLAG=1	001 =			
004F	4304	= 202	ORL A,#BYFNFLAG	001 =			0000
0051	AE	= 203	MOV FLGBYT,A	001 =			
0052	0464	= 204	JMP XMIT	001 =			0000
		= 205 ; 6	ELSE	001 =			
		= 206 ; 7	BYTE FINISHED FLAG:=1	001 =			
		= 207 ; 7	OVERRUN FLAG:=1	001 =			
		= 208	RCV045:	001 =			
		= 209	MOV TVR,A:FLGBYT	001 =			0000
0054	4304	= 210	ORL A,#(BYFNFLAG OR OVRUN)	001 =			0000
0056	AE	= 211	MOV FLGBYT,A	001 =			
		= 212 ; 6	ENDIF	001 =			
		= 213 ; 5	ENDIF	001 =			0000
0057	0464	= 214	JMP XMIT	001 =			

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	370002	032	180	001
		= 215 ; 4	ELSE ; 4 SINCE BYTE FINISHED FLAG=1 THEN ; 2700				
		= 216 ; 5	IF SERIAL INPUT=MARK THEN ; 2802 2700				
0059	265F	= 217 RCV050: JNT0:1 RCV060	VOR ; 30 2700				
		= 218 ; 6	DATA READY FLAG:=1 ; 3000 2700				
005B	4308	= 219 ORL A, #RDYFLG	; 320 2700				
005D	0461	= 220 JMP 23 RCV070	; 34 2700				
		= 221 ; 5	ELSE SINCE SERIAL INPUT=SPACE THEN				
		= 222 ; 6	ERROR FLAG:=1 ; 3500 2700				
005F	4310	= 223 RCV060: ORL A, #ERRFLG	; 37 2700				
		= 224 ; 5	ENDIF ; 39 2700				
		= 225 ; 5	RECEIVE FLAG:=0 ; 40 2700				
		= 226 ; 5	SYNC FLAG:=0 ; 42 2700				
0061	53FC	= 227 RCV070: ANL A, #NOT(SYNFLG OR RCVFLG)	; 4403 1800				
0063	AE	= 228 MOV FLGBYT, A	; 46 1800				
		= 229 ; 4	ENDIF ; 48 1800				
		= 230 ; 3	ENDIF ; 50 1800				
		= 231 ; 2	ENDIF SINCE TRANSMITTING FLAG=1 THEN ; 52 1800				
		= 232 ; 1	ENDIF ; 54 1800				
		= 233 #EJECT	; 56 1800				
		= 234 ;	START OF TRANSMIT ROUTINE ; 58 1800				
		= 235 ;	===== ; 60 1800				
		= 236 ;	===== ; 62 1800				
		= 237 ;	===== ; 64 1800				
		= 238 ; 1	TRANSMITTER OUTPUT BIT IS P2-7 ; 66 1800				
		= 239 ; 1	TICK COUNTER:=TICK COUNTER+1 ; 68 1800				
0064	1C	= 241 XMIT: INC TCKCTR	; 70 1800				
		= 242 ; 1	IF TICK COUNTER MOD 4=0 THEN ; 72 1800				
0065	2303	= 243 MOV DATA, #03H	; 74 1800				
0067	5C	= 244 ANL TCKCTR, A	; 76 1800				
0068	9697	= 245 JNZ RETURN	; 78 1800				
		= 246 ; 2	IF TRANSMITTING FLAG=1 THEN ; 80 1800				
006A	FE	= 247 MOV B, A: FLGBYT	; 82 1800				
006B	37	= 248 CPL A	; 84 1800				
006C	D286	= 249 JB6 XMT040	; 86 1800				
		= 250	IF STPBTS EQ 1 ; 88 1800				
		= 251 ; 3	IF TICK COUNTER=00 1010 00 BINARY THEN ; 90 1800				
		= 252	MOV A, #28H ; 92 1800				
		= 253	XRL A, TCKCTR ; 94 1800				
		= 254	JNZ XMT010 ; 96 1800				
		= 255 ; 4	TRANSMITTING FLAG:=0 ; 98 1800				
		= 256	MOV A, FLGBYT ; 100 1800				
		= 257	ANL A, #NOT TRNGFL ; 102 1800				
		= 258	MOV A, FLGBYT: A2 ; 104 1800				
		= 259	JMP RETURN ; 106 1800				
		= 260	ENDIF ; 108 1800				
		= 261 ; 3	ELSE IF TICK COUNTER=00 1001 00 BINARY THEN ; 110 1800				
006E	2324	= 262 XMT010: MOV A, #24H	; 112 1800				
0070	DC	= 263 XRL A, TCKCTR	; 114 1800				
0071	967B	= 264 JNZ XMT020	; 116 1800				
		= 265 ; 4	SEND END MARK ; 118 1800				
0073	A5	= 266 CLR F1 ; SET FLAG1 TO MARK	; 120 1800				
0074	B5	= 267 CPL F1	; 122 1800				
		= 268	IF STPBTS EQ 0 ; 124 1800				
		= 269 ; 4	TRANSMITTING FLAG:=0 ; 126 1800				

Figure 4 (continued)

LOC	ADDR	DEC	SOURCE STATEMENT	OPCODE	HEX	DISP	DATA
0075	FE	= 270	MOV R0, A, FLGBYT	MOV	00	00	00
0076	53BF	= 271	ANL R1, A, #NOT TRNGFL	ANL	00	00	00
0078	AE	= 272	MOV R0, FLGBYT, A	MOV	00	00	00
0079	0497	= 273	JMP R0, RETURN	JMP	00	00	00
		= 274	ENDIF	ENDIF	00	00	00
		= 275 ; 3	ELSE SINCE TICK COUNTER > THE ABOVE COUNT THEN				
		= 276 ; 4	SEND NEXT BIT				
0078	B822	= 277	XMT020: MOV R0, R0, #XMTBY	MOV	00	00	00
007D	F0	= 278	MOV R0, A, @R0	MOV	00	00	00
007E	67	= 279	RRC A, #1	RRC	00	00	00
007F	A0	= 280	MOV R0, A, @R0, A	MOV	00	00	00
0080	A5	= 281	CLR R0, F1 ; FLAG 1 WILL BE USED TO BUFFER TXD	CLR	00	00	00
0081	E697	= 282	JNC R0, RETURN ; GO TO RETURN POINT IF TXD=SPACE (0)	JNC	00	00	00
0083	B5	= 283	CPL R0, F1 ; ELSE COMPLEMENT FLAG 1 TO A MARK	CPL	00	00	00
0084	0497	= 284	JMP RETURN	JMP	00	00	00
		= 285 ; 3	ENDIF	ENDIF	00	00	00
		= 286 ; 2	ELSE SINCE TRANSMITTING FLAG=0 THEN				
		= 287 ; 3	IF TRANSMIT REQUEST FLAG=1 THEN				
0086	B297	= 288	XMT040: JB5 RETURN	JB5	00	00	00
		= 289 ; 4	XMTBYT:=XMTBYT				
0088	B823	= 290	MOV R0, R0, #XMTBY	MOV	00	00	00
008A	F0	= 291	MOV R0, A, @R0	MOV	00	00	00
008B	B822	= 292	MOV R0, R0, #XMTBY	MOV	00	00	00
008D	A0	= 293	MOV R0, A, @R0, A	MOV	00	00	00
		= 294 ; 4	TRANSMIT REQUEST FLAG:=0				
008E	FE	= 295	MOV R0, A, FLGBYT	MOV	00	00	00
008F	53DF	= 296	ANL R1, A, #NOT TRNGFL	ANL	00	00	00
		= 297 ; 4	TRANSMITTING FLAG:=1				
0091	4340	= 298	ORL R1, A, #TRNGFL	ORL	00	00	00
0093	AE	= 299	MOV R0, FLGBYT, A	MOV	00	00	00
		= 300 ; 4	TICK COUNTER:=0				
0094	BC00	= 301	MOV R0, MTCKCTR, #0	MOV	00	00	00
		= 302 ; 4	SEND SYNC BIT (SPACE)				
0096	A5	= 303	CLR R0, F1 ; SET FLAG 1 TO CAUSE A SPACE	CLR	00	00	00
		= 304 ; 3	ENDIF	ENDIF	00	00	00
		= 305 ; 2	ENDIF	ENDIF	00	00	00
		= 306 ; 1	ENDIF	ENDIF	00	00	00
		= 307	RETURN	RETURN	00	00	00
		= 308 ; 1	RESTORE ACCUMULATOR				
0097	FF	= 309	MOV R0, A, ATEMP	MOV	00	00	00
0098	93	= 310	RETR	RETR	00	00	00
		311	\$EJECT				
		312 ;					
		313 ;	START OF TEST ROUTINE				
		314 ;	=====				
		315 ;					
0100		316	ORG 0100H	ORG	00	00	00
FFFE		317	TMCNT EQU 0100H-2	EQU	00	00	00
001E		318	MFLGBY EQU 0100H-1EH	EQU	00	00	00
001D		319	MSAMCT EQU 0100H-1DH	EQU	00	00	00
001C		320	MTCKCT EQU 0100H-1CH	EQU	00	00	00
		321					
0007		322	ERRCNT EQU R7	EQU	00	00	00
0006		323	PATT EQU 0100H-16H	EQU	00	00	00
		324 ;	TRANSMITTING FLAG:=0				

Figure 4 (continued)

LUC	OBJ	SEQ	SOURCE STATEMENT	INSTR	START	END	LOC	30J
		325 ;						
		326 ;						
		327 ;1 ERROR COUNT:=0						
0100	BF00	328 TEST: MOV	ERRCNT, #0					
		329 ;1 REPEAT						
		330 TLOP:						
		331 ;2 PATTERN:=0						
0102	BE00	332	MOV PATT, #00					
		333 ;2 INITIALIZE TIMER						
0104	23FE	334	MOV A, #TIMCNT					
0106	62	335	MOV T, A					
0107	55	336	STRT T					
0108	25	337	EN TCNTI					
		338 ;2 CLEAR FLAGBYTE						
0109	B81E	339	MOV R0, #NFLGBY					
010B	B000	340	MOV @R0, #0					
		341 ;2 FLAG1=MARK						
010D	A5	342	CLR F1					
010E	B5	343	CPL F1					
		344 ;2 REPEAT						
		345 TLOP:						
		346 ;3 IF TRANSMIT REQUEST FLAG=0 THEN						
010F	B81E	347	MOV R0, #NFLGBY					
0111	F0	348	MOV A, @R0					
0112	B224	349	JB5 TREC					
		350 ;4 NXTBYTE:=PATTERN						
0114	B923	351	MOV R1, #NXTBY					
0116	FE	352	MOV A, PATT					
0117	A1	353	MOV @R1, A					
		354 ;4 TRANSMIT REQUEST FLAG=1						
0118	35	355	DIS TCNTI ; LOCK OUT TIMER INTERRUPT					
		356	; SO THAT MUTUAL EXCLUSION IS MAINTAINED WHILE					
		357	; THE FLAG BYTE IS BEING MODIFIED					
0119	F0	358	MOV A, @R0					
011A	4320	359	ORL A, #TRRQFL					
011C	A0	360	MOV @R0, A					
011D	25	361	EN TCNTI					
011E	1622	362	JTF TESTA					
0120	2424	363	JMP TREC					
0122	140A	364 TESTA: CALL	UART ; CALL UART BECAUSE TIMER OVERFLOWED DURING LOCKOUT					
		365 ;3 ENDF						
		366 ;3 IF DATA READY FLAG=1 THEN						
		367 TREC:						
0124	F0	368	MOV A, @R0					
0125	37	369	CPL A					
0126	7238	370	JB3 TREC					
		371 ;4 PATTERN:=OKDATA						
0128	B920	372	MOV R1, #OKDAT					
012A	F1	373	MOV A, @R1					
012B	AE	374	MOV PATT, A					
		375 ;4 DATA READY FLAG:=0						
012C	35	376	DIS TCNTI ; LOCK OUT TIMER INTERRUPT					
		377	; SO THAT MUTUAL EXCLUSION IS MAINTAINED WHILE					
		378	; THE FLAG BYTE IS BEING MODIFIED					
012D	F0	379	MOV A, @R0					

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	THINKSTATE	30002	002	100	001
012E 53F7		380	ANL A, #NOT DRDYFL					
0130 A0		381	MOV @R0, A					
0131 25		382	EN TONTI					
0132 1636		383	JTF TESTB					
0134 2438		384	JMP TRECE					
0136 140A		385	TESTB: CALL UART ; CALL UART IF TIMER OVERFLOWED DURING LOCKOUT					
		386	TRECE:					
		387 ; 3	ENDIF					
		388 ; 2	UNTIL ERROR FLAG OR OVERRUN FLAG					
0138 F0		389	MOV A, @R0					
0139 5390		390	ANL A, #(OVRUN OR ERRFLG)					
013B C60F		391	JZ TILOP					
		392 ; 2	INCREMENT ERROR COUNT					
013D 1F		393	INC ERRCNT					
		394 ; 1	UNTIL FOREVER					
013E 2402		395	JMP TLOP					
		396 : EOF						
		397	END					
USER SYMBOLS								
ATEMP 0007	BYFNFL 0004	DRDYFL 0008	ERRCNT 0007	ERRFLG 0010	FLGBYT 0006	MARK 0080	MDATA 0021	
MFLGBY 001E	MXNTBY 0023	MOKDAT 0020	MSAMCT 001D	MTCKCT 001C	MXMTBY 0022	OMARK 0015	OSPACE 0011	
OVRUN 0080	PATT 0006	RCV000 0017	KCV010 0024	RCV020 0033	RCV030 0038	RCV040 0042	KCV045 0054	
RCV050 0059	RCV060 005F	RCV070 0061	RCVFLG 0001	REG0 0000	RETURN 0097	SAMCTR 0005	SPACE FF7F	
STPBT5 0000	SYNFLG 0002	TCKCTR 0004	TEST 0100	TESTA 0122	TESTB 0136	TILOP 010F	TINCNT FF7E	
TISR 0007	TLOP 0102	TREC 0124	TRECE 0138	TRNGFL 0040	TRRGFL 0020	UART 000A	XMI1 0064	
XMT010 006E	XMT020 007B	XMT040 0086						

ASSEMBLY COMPLETE. NO ERRORS

Figure 4 (continued)

All mnemonics copyrighted © Intel Corporation 1979.

MULTIPLY ALGORITHMS

Most microcomputer programmers have at one time or another implemented a multiply routine as part of a larger program. The usual procedure is to find an algorithm that works and modify it to work on the machine being used. There is nothing wrong with this approach. If engineers felt that they had to reinvent the wheel every time a new design is undertaken, that's probably what most of us would be doing—designing wheels. If the efficiency of the multiply algorithm, either in terms

of code size or execution time is important, however, it is necessary to be reasonably familiar with the multiplication process so that appropriate optimizations for the machine being used can be made.

To understand how multiplication operates in the binary number system, consider the multiplication of two four bit operands A and B. The "ones and zeros" in A and B represent the coefficients of two polynomials. The operation $A \times B$ can be represented as the following multiplication of polynomials:

$$\begin{array}{r}
 \begin{array}{cccc}
 A3 \cdot 2^3 & + & A2 \cdot 2^2 & + & A1 \cdot 2^1 & + & A0 \cdot 2^0 \\
 \times \begin{array}{cccc}
 B3 \cdot 2^3 & + & B2 \cdot 2^2 & + & B1 \cdot 2^1 & + & B0 \cdot 2^0
 \end{array} \\
 \hline
 & & + & B0A3 \cdot 2^3 & + & B0A2 \cdot 2^2 & + & B0A1 \cdot 2^1 & + & B0A0 \cdot 2^0 \\
 + & B1A3 \cdot 2^4 & + & B1A2 \cdot 2^3 & + & B1A1 \cdot 2^2 & + & B1A0 \cdot 2^1 \\
 + & B2A3 \cdot 2^5 & + & B2A2 \cdot 2^4 & + & B2A1 \cdot 2^3 & + & B2A0 \cdot 2^2 \\
 + & B3A3 \cdot 2^6 & + & B3A2 \cdot 2^5 & + & B3A1 \cdot 2^4 & + & B3A0 \cdot 2^3
 \end{array}
 \end{array}$$

The sum of all these terms represents the product of A and B. The simplest multiply algorithm factors the above terms as follows:

$$A * B = B0 * (A) * 2^0 + B1 * (A) * 2^1 + B2 * (A) * 2^2 + B3 * (A) * 2^3$$

Since the coefficients of B (i.e., B0, B1, B2, and B3) can only take on the binary values of 1 or 0, the sum of the products can be formed by a series of simple adds and multiplications by two. The simplest implementation of this would be:

```
MULTIPLY:
  PRODUCT = 0
  IF B0 = 1 THEN PRODUCT = PRODUCT + A
  IF B1 = 1 THEN PRODUCT = PRODUCT + 2 * A
  IF B2 = 1 THEN PRODUCT = PRODUCT + 4 * A
  IF B3 = 1 THEN PRODUCT = PRODUCT + 8 * A
END MULTIPLY
```

In order to conserve memory, the above straight line code is normally converted to the following loop:

```
MULTIPLY:
  PRODUCT = 0
  COUNT = 4
  REPEAT
    IF B[0] = 1 THEN PRODUCT = PRODUCT + A ENDIF
    A = 2 * A
    B = B/2
    COUNT = COUNT - 1
  UNTIL COUNT = 0
END MULTIPLY
```

The repeated multiplication of A by two (which can be performed by a simple left shift) forms the terms 2*A, 4*A, and 8*A. The variable B is divided by two (performed by a simple right shift) so that the least significant bit can always be used to determine whether the addition should be executed during each pass through the loop. It is from these shifting and addition opera-

tions that the "shift and add" algorithm takes its common name.

The "shift and add" algorithm shown above has two areas where efficiency will be lost if implemented in the manner shown. The first problem is that the addition to the partial product is double precision relative to the two operands. The other problem, which is also related to double precision operations, is that the A operand is double precision and that it must be left shifted and then the B operand must be right shifted. An examination of the "longhand" polynomial multiplication will reveal that, although the partial product is indeed double precision, each addition performed is only single precision. It would be desirable to be able to shift the partial product as it is formed so that only single precision additions are performed. This would be especially true if the partial product could be shifted into the "B" operand since one bit of the partial product is formed during each pass through the loop and (happily) one bit of the "B" operand is vacated. To do this, however, it is necessary to modify the algorithm so that both of the shifts that occur are of the same type.

To see how this can be done one can take the basic multiplication equation already presented:

$$A * B = B0 * (A * 2^0) + B1 * (A * 2^1) + B2 * (A * 2^2) + B3 * (A * 2^3)$$

and factoring 2^4 from the right side:

$$A * B = 2^4 [B0 * (A * 2^{-4}) + B1 * (A * 2^{-3}) + B2 * (A * 2^{-2}) + B3 * (A * 2^{-1})]$$

This operation has resulted in a term (within the brackets) which can be formed by right shifts and adds and then multiplied by 2^4 to get the final result. The resulting algorithm, expanded to form an eight by eight multiplication, is shown in figure 5. Note that although the result is a full sixteen bits, the algorithm only performs eight bit additions and that only a single sixteen bit shift operation is involved. This has the effect of reducing both the code space and the execution time for the routine.

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE(:F1.MPVS.HED)
		3	*****
		4	;
		5	;
		6	MPV8X8
		7	*****
		8	;
		9	;
		10	THIS UTILITY PROVIDES AN 8 BY 8 UNSIGNED MULTIPLY
		11	AT ENTRY:
		12	A = LOWER EIGHT BITS OF DESTINATION OPERAND
		13	XA= DON'T CARE
		14	R1= POINTER TO SOURCE OPERAND (MULTIPLIER) IN INTERNAL MEMORY

Figure 5


```

LOC OBJ SEQ SOURCE STATEMENT
= 68 MPY8A
= 69 ; 3 MULTIPLICAND:=MULTIPLICAND*15-8;MULTIPLIER
000E 2A = 70 XCH A,XA
000F 61 = 71 ADD A,@R1
0010 67 = 72 RRC A
0011 2A = 73 XCH A,XA
0012 67 = 74 RRC A
0013 E804 = 75 DJNZ COUNT,MPY8LP
0015 83 = 76 RET
= 77 ; 3 MULTIPLICAND:=MULTIPLICAND/2
= 78 ; 2 ENDIF
= 79 ; 2 COUNT:=COUNT-1
= 80 ; 1 UNTIL COUNT=0
= 81 ; 1 END MPY8X8
= 82 END

```

```

USER SYMBOLS
COUNT 0003 DIGPR 0003 ICNT 0004 MPY8A 000E MPY8LP 0004 MPY8X8 0000 XA 0002

```

ASSEMBLY COMPLETE. NO ERRORS

All mnemonics copyrighted © Intel Corporation 1979.

DIVIDE ALGORITHMS

In order to understand binary division a four bit operation will again be used as an example. The following algorithm will perform a four by four division:

```

DIVIDE:
IF 16*DIVISOR>= DIVIDEND THEN
SET OVERFLOW ERROR FLAG
ELSE
IF 8*DIVISOR>= DIVIDEND THEN
QUOTIENT[3]=1
DIVIDEND:=DIVIDEND-8*DIVISOR
ELSE
QUOTIENT[3]=0
ENDIF
IF 4*DIVISOR>= DIVIDEND THEN
QUOTIENT[2]=1
DIVIDEND:=DIVIDEND-4*DIVISOR
ELSE
QUOTIENT[2]=0
ENDIF
IF 2*DIVISOR>= DIVIDEND THEN
QUOTIENT[1]=1
DIVIDEND:=DIVIDEND-2*DIVISOR
ELSE
QUOTIENT[1]=0
ENDIF
IF 1*DIVISOR>= DIVIDEND THEN
QUOTIENT[0]=1
DIVIDEND:=DIVIDEND-1*DIVISOR
ELSE
QUOTIENT[0]=0
ENDIF
ENDIF
END DIVIDE

```

The algorithm is easy to understand. The first test asks if the division will fit into the dividend sixteen times. If it will, the quotient cannot be expressed in only four bits so an overflow error flag is set and the divide algorithm ends. The algorithm then proceeds to determine if eight times the divisor fits, four times, etc. After each test it either sets or clears the appropriate quotient bit and modifies the dividend. To see this algorithm in action, consider the division of 15 by 5:

00001111	(15)	
- 01010000	(16*5)	
<hr/>		Doesn't fit—no overflow
00001111	(15)	
- 00101000	(8*5)	
<hr/>		Doesn't fit—Q[3]=0
00001111	(15)	
- 00010100	(4*5)	
<hr/>		Doesn't fit—Q[2]=0
00001111	(15)	
- 00001010	(2*5)	
<hr/>		Fits—Q[1]=1
00000101	(15-2*5)	
- 00000101	(1*5)	
<hr/>		Fits—Q[0]=1
00000000		

The result is Q = 0011 which is the binary equivalent of 3—the correct answer. Clearly this algorithm can (and has been) converted to a loop and used to perform divisions. An examination of the procedure, however, will show that it has the same problems as the original multiplication algorithm.

The first problem is that double precision operations are involved with both the comparison of the division with the dividend and the conditional subtraction. The second problem is that as the quotient bits are derived they must be shifted into a register. In order to reduce the register requirements, it would be desirable to shift them into the divisor register as they are generated since the divisor register gets shifted anyway. Unfortunately the quotient bits are derived most significant bits first so doing this will form a mirror image of the quotient—not very useful.

Both of these problems can be solved by observing that the algorithm presented for divide will still work if both sides of all the "equations" involving the dividend are divided by sixteen. The looping algorithm then would proceed as follows:

```
DIVIDE:
QUOTIENT:= 0
COUNT:= 4
DIVIDEND:= DIVIDEND/16
IF DIVISOR>= DIVIDEND THEN
OVERFLOW FLAG:= 1
ELSE
REPEAT
DIVIDEND:= DIVIDEND*2
QUOTIENT:= QUOTIENT*2
IF DIVISOR>= DIVIDEND THEN
QUOTIENT:= QUOTIENT + 1/2 SET QUOTIENT[0]*/
DIVIDEND:= DIVIDEND - DIVISOR
ENDIF
COUNT:= COUNT - 1
UNTIL COUNT= 0
ENDIF
END DIVIDE
```

When this algorithm is implemented on a computer which does not have a direct compare instruction the comparison is done by subtraction and the inner loop of the algorithm is modified as follows:

```
REPEAT
DIVIDEND:= DIVIDEND*2
QUOTIENT:= QUOTIENT*2
DIVIDEND:= DIVIDEND - DIVISOR
IF BORROW= 0 THEN
QUOTIENT:= QUOTIENT + 1
ELSE
DIVIDEND:= DIVIDEND + DIVISOR
ENDIF
COUNT:= COUNT - 1
UNTIL COUNT= 0
```

An implementation of this algorithm using the 8049 instruction set is shown in figure 6. This routine does an unsigned divide of a 16 bit quantity by an eight bit quantity. Since the multiply algorithm of figure 5 generates a 16 bit result from the multiplication of two eight bit operands, these two routines complement each other and can be used as part of more complex computations.

1515-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE(X:F1:DIV16.HED)
		3	*****
		4	*****
		5	*****
		6	*****
		7	*****
		8	*****
		9	*****
		10	*****
		11	*****
		12	*****
		13	*****
		14	*****
		15	*****
		16	*****
		17	*****

1 \$MACROFILE

2 \$INCLUDE(X:F1:DIV16.HED)

3 *****

4 *****

5 *****

6 *****

7 *****

8 *****

9 *****

10 *****

11 *****

12 *****

13 *****

14 *****

15 *****

16 *****

17 *****

THIS UTILITY PROVIDES AN 16 BY 8 UNSIGNED DIVIDE

AT ENTRY:

A = LOWER EIGHT BITS OF DESTINATION OPERAND

XA = UPPER EIGHT BITS OF DIVIDEND

R1= POINTER TO DIVISOR IN INTERNAL MEMORY

AT EXIT:

A = LOWER EIGHT BITS OF RESULT

XA= REMAINDER

Figure 6

LOC	OBJ	SEQ	SOURCE STATEMENT	THAT STATE	092	180	301
		= 18 :*	C = SET IF OVERFLOW ELSE CLEARED	THAT TOUO	7	AT	*
		= 19 :*			7	AT	*
		= 20 :*****					
		= 21 :					
		= 22 :					
		= 22 :#INCLUDE(F1:DIV16 PDL)					
		= 24 :1 DIV16					
		= 25 :1 COUNT:=8					
		= 26 :1 DIVIDEND(15-8):=DIVIDEND(15-8)-DIVISOR					
		= 27 :1 IF BORROW=0 THEN /* IT FITS*/					
		= 28 :2 SET OVERFLOW FLAG					
		= 29 :1 ELSE					
		= 30 :2 RESTORE DIVIDEND					
		= 31 :2 REPEAT					
		= 32 :3 DIVIDEND:=DIVIDEND*2					
		= 33 :3 QUOTIENT:=QUOTIENT*2					
		= 34 :3 DIVIDEND(15-8):=DIVIDEND(15-8)-DIVISOR					
		= 35 :3 IF BORROW=1 THEN					
		= 36 :4 RESTORE DIVIDEND					
		= 37 :3 ELSE					
		= 38 :4 QUOTIENT(0)=1					
		= 39 :3 ENDIF					
		= 40 :3 COUNT=COUNT-1					
		= 41 :2 UNTIL COUNT=0					
		= 42 :2 CLEAR OVERFLOW FLAG					
		= 43 :1 ENDIF					
		= 44 :1 ENDDIVIDE					
		= 45 :					
		= 46 :EQUATES					
		= 47 :*****					
		= 48 :					
0002		= 49 :XA EQU R2					
0003		= 50 :COUNT EQU R3					
		= 51 :					
		= 52 :\$EJECT					
		= 53 :#INCLUDE(F1:DIV16)					
		= 54 :1 DIV16:					
0000 2A		= 55 :DIV16: XCH A,XA	ROUTINE WORKS MOSTLY WITH BITS 15-8				
		= 56 :1 COUNT:=8					
0001 8B08		= 57 :MOV COUNT,#8					
		= 58 :1 DIVIDEND(15-8):=DIVIDEND(15-8)-DIVISOR					
0003 37		= 59 :CPL A					
0004 61		= 60 :ADD A,@R1					
0005 37		= 61 :CPL A					
		= 62 :1 IF BORROW=0 THEN /* IT FITS*/					
0006 F60B		= 63 :JC DIVIA					
		= 64 :2 SET OVERFLOW FLAG					
0008 A7		= 65 :CPL C					
0009 0424		= 66 :JMP DIVIB					
		= 67 :1 ELSE					
		= 68 :DIVIA:					
		= 69 :2 RESTORE DIVIDEND					
000B 61		= 70 :ADD A,@R1					
		= 71 :2 REPEAT					
		= 72 :DIVILP:					
		= 73 :3 DIVIDEND:=DIVIDEND*2					

Figure 6 (continued)

```

      = 74 :3      QUOTIENT=QUOTIENT*2
0000 97      = 75      CLR      C
0000 2A      = 76      XCH      A,XA
000E F7      = 77      PLC      A
000F 2A      = 78      XCH      A,XA
0010 F7      = 79      PLC      A
0011 E618    = 80      JNC      DIVIE
0013 37      = 81      CPL      A
0014 61      = 82      ADD      A,@R1
0015 37      = 83      CPL      A
0016 0420    = 84      JMP      DIVIC
      = 85 :2      DIVIDEND(15-8)=DIVIDEND(15-8)-DIVISOR
0018 37      = 86 DIVIE: CPL      A
0019 61      = 87      ADD      A,@R1
001A 37      = 88      CPL      A
      = 89 :3      IF BORROW=1 THEN
001B E620    = 90      JNC      DIVIC
      = 91 :4      RESTORE DIVIDEND
001D 61      = 92      ADD      A,@R1
001E 0421    = 93      JMP      DIVID
      = 94 :3      ELSE
      = 95 DIVIC:
      = 96 :4      QUOTIENT(0)=1
0020 1A      = 97      INC      XA
      = 98 :3      ENDIF
      = 99 :3      COUNT=COUNT-1
      = 100 :2     UNTIL COUNT=0
0021 E80C    = 101 DIVID: DJNZ    COUNT,DIVILP
      = 102 :2     CLEAR OVERFLOW FLAG
0023 97      = 103      CLR      C
      = 104 :1     ENDIF
      = 105 :1     ENDDIVIDE
0024 2B      = 106 DIVIS: XCH      A,XA
0025 83      = 107      RET
      = 108 END

USER SYMBOLS
COUNT 0003  DIV16 0000  DIV1A 0006  DIV1B 0024  DIVIC 0020  DIVID 0021  DIVIE 0018  DIVILP 000C
XA      0002

ASSEMBLY COMPLETE, NO ERRORS

```

Figure 6 (continued)

All mnemonics copyrighted © Intel Corporation 1979.

BINARY AND BCD CONVERSIONS

The conversion of a binary value to a BCD (binary coded decimal) number can be done with a very straightforward algorithm:

```

CONVERT_TO_BCD:
  BCDACCUM:=0
  COUNT:=PRECISION
  REPEAT
    BIN:=BIN * 2
    BCD:=BCD * 2 + CARRY
    COUNT:=COUNT - 1
  UNTIL COUNT=0
END CONVERT_TO_BCD

```

The variable **BCDACCUM** is a BCD string used to accumulate the result; the variable **BIN** is the binary number to be converted. **PRECISION** is a constant which gives the length, in binary bits of **BIN**. To see how this works, assume that **BIN** is a sixteen bit value with the most significant bit set. On the first pass through the loop the multiplication of **BIN** will result in a carry and this carry will be added to BCD. On the remaining passes through the loop BCD will be multiplied by two 15 times. The initial carry into BCD will be multiplied by 2^{15} or 32678, which is the "value" of the most significant bit of **BIN**. The process repeats with each bit of **BIN** being introduced to **BCDACCUM** and then being scaled up on successive passes through the loop. Figure 7 shows the implementation of this algorithm for the 8049.

IS15-11 MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	*\$MACROFILE
		2	*\$INCLUDE(.F1.CONBCD.HED)
		3	*****
		4	;
		5	;* CONBCD
		6	;
		7	*****
		8	;
		9	;* THIS UTILITY CONVERTS A 16 BIT BINARY VALUE TO BCD
		10	;* AT ENTRY
		11	;* A = LOWER EIGHT BITS OF BINARY VALUE
		12	;* XA= UPPER EIGHT BITS OF BINARY VALUE
		13	;* R0= POINTER TO A PACKED BCD STRING
		14	;
		15	;* AT EXIT
		16	;* A = UNDEFINED
		17	;* XA= UNDEFINED
		18	;* C = SET IF OVERFLOW ELSE CLEARED
		19	;
		20	*****
		21	;
		22	;
		23	*\$INCLUDE(.F1.CONBCD.PDL)
		24	1 CONVERT_TO_BCD
		25	1 BCDACC:=0
		26	1 COUNT =16
		27	1 REPEAT
		28	2 BIN:=BIN*2
		29	2 BCD:=BCD*2+CARRY
		30	2 IF CARRY FROM BCDACC GOTO ERROR EXIT
		31	2 COUNT:=COUNT-1
		32	1 UNTIL COUNT=0
		33	1 END CONVERT_TO_BCD
		34	;
		35	;
		36	*****
		37	;
0002		38	XA EQU R2
0003		39	COUNT EQU R3
0004		40	ICNT EQU R4
		41	;
0003		42	DIGPR EQU 3
		43	;
		44	\$EJECT
		45	*\$INCLUDE(.F1.CONBCD)
		46	;
0005		47	TEMP1 SET R5
		48	;
		49	1 CONVERT_TO_BCD
		50	CNBCD
		51	1 BCDACC:=0
0000 28		52	XCH A,R0

Figure 7

LOC	OBJ	SEQ	SOURCE STATEMENT						
0001	A0	= 53	MOV R1, A						
0002	28	= 54	XCH A, R0						
0003	BC03	= 55	MOV ICNT, #DIGPR						
0005	B100	= 56	BCDCOR MOV @R1, #00						
0007	13	= 57	INC R1						
0008	EC05	= 58	DJNZ ICNT, BCDCOR						
		= 59	:1 COUNT =16						
000A	BB10	= 60	MOV COUNT, #16						
		= 61	:1 REPEAT						
		= 62	BCDCOB						
		= 63	:2 BIN:=BIN*2						
000C	97	= 64	CLR C						
000D	F7	= 65	RLC A						
000E	2A	= 66	XCH A, XA						
000F	F7	= 67	RLC A						
0010	2A	= 68	XCH A, XA						
		= 69	:2 BCD:=BCD*2+CARRY						
0011	28	= 70	XCH A, R0						
0012	A9	= 71	MOV R1, A						
0013	28	= 72	XCH A, R0						
0014	BC03	= 73	MOV ICNT, #DIGPR						
0016	A0	= 74	MOV TEMP1, A						
0017	F1	= 75	BCDOC MOV A, @R1						
0018	71	= 76	ADDC A, @R1						
0019	57	= 77	DA A						
001A	A1	= 78	MOV @R1, A						
001B	19	= 79	INC R1						
001C	EC17	= 80	DJNZ ICNT, BCDOC						
001E	FD	= 81	MOV A, TEMP1						
		= 82	:2 IF CARRY FROM BCDACC GOTO ERROR EXIT						
001F	F624	= 83	JC BCDCOB						
		= 84	:2 COUNT =COUNT-1						
		= 85	:1 UNTIL COUNT=0						
0021	EB0C	= 86	DJNZ COUNT, BCDCOB						
0023	97	= 87	CLR C ; CLEAR CARRY TO INDICATE NORMAL TERMINATION						
		= 88	:1 END CONVERT_TO_BCD						
0024	83	= 89	BCDCOB RET						
		= 90	END						

USER SYMBOLS

BCDCOR 0005	BCDCOB 000C	BCDCOD 0024	BCDOC 0017	CNBD 0000	COUNT 0003	DIGPR 0003	ICNT 0004
TEMP1 0005	XA 0002						

ASSEMBLY COMPLETE, NO ERRORS

Figure 7 (continued)

The conversion of a BCD value to binary is essentially the same process as converting a binary value to BCD.

```

CONVERT_TO_BINARY
  BIN:=0
  COUNT:=DIGNO
  REPEAT
    BCDACCUM:=BCDACCUM*10
    BIN:=10*BIN+CARRY DIGIT
    COUNT:=COUNT-1
  UNTIL COUNT=0
END CONVERT_TO_BINARY

```

The only complexity is the two multiplications by ten. The BCDACCUM can be multiplied by ten by shifting it left four places (one digit). The variable BIN could be multiplied using the multiply algorithm already discussed, but it is usually more efficient to do this by mak-

ing the following substitution:

$$BIN = 10 * BIN = (2) * (5) * (BIN) = 2 * (2 * 2 + 1) * BIN$$

This implies that the value $10 * BIN$ can be generated by saving the value of BIN and then shifting BIN two places left. After this the original value of BIN can be added to the new value of BIN (forming $5 * BIN$) and then BIN can be multiplied by two. It is often possible to implement the multiplication of a value by a constant by using such techniques. Figure 8 shows an 8049 routine which converts BCD values to binary. This routine differs slightly from the algorithm above in that the BCD digits are read, and converted to binary, two digits at a time. Protection has also been added to detect BCD operands which, if converted, would yield binary values beyond the range of the result.

1

ISIS-11 MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE(F1:CONBIN.HED)
		3	*****
		4	;
		5	CONBIN
		6	;
		7	*****
		8	;
		9	THIS UTILITY CONVERTS A 6 DIGIT BCD VALUE TO BINARY
		10	AT ENTRY:
		11	R0= POINTER TO A PACKED BCD STRING
		12	;
		13	AT EXIT:
		14	A = LOWER EIGHT BITS OF THE BINARY RESULT
		15	XA= UPPER EIGHT BITS OF THE BINARY RESULT
		16	C = SET IF OVERFLOW ELSE CLEARED
		17	;
		18	*****
		19	;
		20	;
		21	\$INCLUDE(F1:CONBIN.PDL)
		22	;
		23	;
		24	1 CONVERT_TO_BINARY
		25	1 POINTER0:=POINTER0+DIGITPAIR-1
		26	1 COUNT:=DIGITPAIR
		27	1 BIN:=0
		28	1 REPEAT
		29	2 BIN:=BIN*10
		30	2 BIN:=BIN+MEM(R0)[7-4]
		31	2 BIN:=BIN*10
		32	2 BIN:=BIN+MEM(R0)[3-0]

```

= 34 : 2 COUNT:=COUNT-1
= 35 : 1 UNTIL COUNT=0
= 36 : 1 END CONVERT_TO_BINARY
= 37 :
= 38 : EQUATES
= 39 :
= 40 :
= 41 : 0002 EQU R2
= 42 : 0003 EQU R3
= 43 : 0004 EQU R4
= 44 :
= 45 : 0003 EQU 3
= 46 :
= 47 : $EJECT
= 48 : $INCLUDE< F1.CONBIN>
= 49 :
= 50 : TEMP1 SET R5
= 51 : TEMP2 SET R6
= 52 :
= 53 : 1 CONVERT_TO_BINARY
= 54 : CONBIN:
= 55 : 1 POINTER0:=POINTER0+DIGITPAIR-1
= 56 : MOV A,R0
= 57 : ADD A,#DIGPR-1
= 58 : MOV R0,A
= 59 : 1 COUNT:=DIGITPAIR
= 60 : MOV COUNT,#DIGPR
= 61 : 1 BIN=0
= 62 : CLR A
= 63 : MOV XA,A
= 64 : 1 REPEAT
= 65 : CONBLP:
= 66 : 2 BIN:=BIN*10
= 67 : CALL CONB10
= 68 : JC CONBER
= 69 : 2 BIN:=BIN+MEM(R0)[7-4]
= 70 : MOV TEMP1,A
= 71 : MOV A,R0
= 72 : SWAP A
= 73 : ANL A,#0FH
= 74 : ADD A,TEMP1
= 75 : XCH A,XA
= 76 : ADDC A,#00
= 77 : XCH A,XA
= 78 : JC CONBER
= 79 : 2 BIN:=BIN*10
= 80 : CALL CONB10
= 81 : JC CONBER
= 82 : 2 BIN:=BIN+MEM(R0)[3-0]
= 83 : MOV TEMP1,A
= 84 : MOV A,R0
= 85 : ANL A,#0FH
= 86 : ADD A,TEMP1
= 87 : XCH A,XA

```

```

0005 = 50 TEMP1 SET R5
0006 = 51 TEMP2 SET R6
= 52 :
= 53 : 1 CONVERT_TO_BINARY
= 54 : CONBIN:
= 55 : 1 POINTER0:=POINTER0+DIGITPAIR-1
0000 F8 = 56 MOV A,R0
0001 0302 = 57 ADD A,#DIGPR-1
0003 A8 = 58 MOV R0,A
= 59 : 1 COUNT:=DIGITPAIR
0004 B803 = 60 MOV COUNT,#DIGPR
= 61 : 1 BIN=0
0006 27 = 62 CLR A
0007 AA = 63 MOV XA,A
= 64 : 1 REPEAT
= 65 : CONBLP:
= 66 : 2 BIN:=BIN*10
0008 142B = 67 CALL CONB10
000A F62A = 68 JC CONBER
= 69 : 2 BIN:=BIN+MEM(R0)[7-4]
000C AD = 70 MOV TEMP1,A
000D F0 = 71 MOV A,R0
000E 47 = 72 SWAP A
000F 530F = 73 ANL A,#0FH
0011 6D = 74 ADD A,TEMP1
0012 2A = 75 XCH A,XA
0013 1300 = 76 ADDC A,#00
0015 2A = 77 XCH A,XA
0016 F62A = 78 JC CONBER
= 79 : 2 BIN:=BIN*10
0018 142B = 80 CALL CONB10
001A F62A = 81 JC CONBER
= 82 : 2 BIN:=BIN+MEM(R0)[3-0]
001C AD = 83 MOV TEMP1,A
001D F0 = 84 MOV A,R0
001E 530F = 85 ANL A,#0FH
0020 6D = 86 ADD A,TEMP1
0021 2A = 87 XCH A,XA

```


LOC	OBJ	SEQ	SOURCE STATEMENT
0022	1300	= 88	ADDC A, #00
0024	2A	= 89	XCH A, XA
0025	F62A	= 90	JC CONBER
		= 91 ; 2	POINTER0:=POINTER0-1
0027	C8	= 92	DEC R0
		= 93 ; 2	COUNT:=COUNT-1
		= 94 ; 1	UNTIL COUNT=0
0028	EB08	= 95	DJNZ COUNT, CONBLP
		= 96 ; 1	END CONVERT_TO_BINARY
002A	83	= 97	CONBER: RET
		= 98	REJECT
		= 99	
		= 100	
		= 101	UTILITY TO MULTIPLY BIN BY 10
		= 102	CARRY WILL BE SET IF OVERFLOW OCCURS
		= 103	
002B	AD	= 104	CONB10: MOV TEMP1, A ; SAVE A
002C	2A	= 105	XCH A, XA ; SAVE XA
002D	AE	= 106	MOV TEMP2, A
002E	2A	= 107	XCH A, XA
		= 108 ;	
002F	97	= 109	CLR C
0030	F7	= 110	PLC A ; BIN:=BIN*2
0031	2A	= 111	XCH A, XA
0032	F7	= 112	RLC A
0033	2A	= 113	XCH A, XA
0034	F646	= 114	JC CONB1E ; ERROR ON OVERFLOW
		= 115 ;	
0036	F7	= 116	PLC A ; BIN:=BIN*4
0037	2A	= 117	XCH A, XA
0038	F7	= 118	RLC A
0039	2A	= 119	XCH A, XA
003A	F646	= 120	JC CONB1E ; ERROR ON OVERFLOW
		= 121 ;	
003C	6D	= 122	ADD A, TEMP1 ; BIN:=BIN*5
003D	2A	= 123	XCH A, XA
003E	7E	= 124	ADDC A, TEMP2
003F	2A	= 125	XCH A, XA
0040	F646	= 126	JC CONB1E ; ERROR ON OVERFLOW
		= 127 ;	
0042	F7	= 128	RLC A ; BIN:=BIN*10
0043	2A	= 129	XCH A, XA
0044	F7	= 130	RLC A
0045	2A	= 131	XCH A, XA
		= 132 ;	
0046	83	= 133	CONB1E: RET
		= 134	
		= 135 ;	
		= 136	END

USER SYMBOLS

CONB10	002B	CONB1E	0046	CONBER	002A	CONBIN	0000	CONBLP	0008	COUNT	0003	DIGPR	0003	ICNT	0004
TEMP1	0005	TEMP2	0006	XA	0002										

ASSEMBLY COMPLETE, NO ERRORS

CONCLUSION

The design goals of the full duplex serial communication software were realized. It was found that the software was able to maintain a throughput of 45 percent of the available time of the 8048. This implies that an 8048 running full duplex serial link will still outperform earlier members of the family running without the serial I/O requirement. It is also possible to run this program in an 8048 or 8148 at 1500 baud with the same 45 percent CPU utilization.

The execution times for the other routines that have been discussed have been summarized in Table 1. All of these routines were written to maintain maximum use of the 8048. The resulting execution times and code size are those that one can expect to see in a real application. The results that were obtained clearly show the efficiency and speed of the 8048. The equivalent of the 8048 are also shown. It is clear that the 8048 represents a substantial performance advantage over the 8048. Considering in most applications that the 8048 is

1

CONCLUSION

The design goals of the full duplex serial communications software were realized; if transmission and reception are occurring concurrently, only 42 percent of the real time available to the 8049 will be consumed by the serial link. This implies that an 8049 running full duplex serial I/O will still outperform earlier members of the family running without the serial I/O requirement. It is also possible to run this program in an 8048 or 8748 at 1200 baud with the same 42 percent CPU utilization.

The execution times for the other routines that have been discussed have been summarized in Table 1. All of these routines were written to maintain maximum usability rather than minimum code size or execution time. The resulting execution times and code size are therefore what the user can expect to see in a real application. The results that were obtained clearly show the efficiency and speed of the 8049. The equivalent times for the 8048 are also shown. It is clear that the 8049 represents a substantial performance advantage over the 8048. Considering, in most applications, that the 8048 is

the highest performance microcomputer available to date, the performance advantage of the 8049 should allow the cost benefits of a single chip microcomputer to be realized in many applications which up until now have required too much "computer power" for a single chip approach.

	EXECUTION TIME (MICROSECONDS)		
	BYTES	8049	8048
MPY8	21	109	200
DIV 16	37	183 MIN 204 MAX	335 MIN 375 MAX
CONBCD	36	733	1348
CONBIN	70	388	713

Table 1. Program Performance

TEST 1	NOV	104	COMB	NOV	80	8048
TEST 2	NOV	104	COMB	NOV	80	8048
TEST 3	NOV	104	COMB	NOV	80	8048
TEST 4	NOV	104	COMB	NOV	80	8048
TEST 5	NOV	104	COMB	NOV	80	8048
TEST 6	NOV	104	COMB	NOV	80	8048
TEST 7	NOV	104	COMB	NOV	80	8048
TEST 8	NOV	104	COMB	NOV	80	8048
TEST 9	NOV	104	COMB	NOV	80	8048
TEST 10	NOV	104	COMB	NOV	80	8048
TEST 11	NOV	104	COMB	NOV	80	8048
TEST 12	NOV	104	COMB	NOV	80	8048
TEST 13	NOV	104	COMB	NOV	80	8048
TEST 14	NOV	104	COMB	NOV	80	8048
TEST 15	NOV	104	COMB	NOV	80	8048
TEST 16	NOV	104	COMB	NOV	80	8048
TEST 17	NOV	104	COMB	NOV	80	8048
TEST 18	NOV	104	COMB	NOV	80	8048
TEST 19	NOV	104	COMB	NOV	80	8048
TEST 20	NOV	104	COMB	NOV	80	8048
TEST 21	NOV	104	COMB	NOV	80	8048
TEST 22	NOV	104	COMB	NOV	80	8048
TEST 23	NOV	104	COMB	NOV	80	8048
TEST 24	NOV	104	COMB	NOV	80	8048
TEST 25	NOV	104	COMB	NOV	80	8048
TEST 26	NOV	104	COMB	NOV	80	8048
TEST 27	NOV	104	COMB	NOV	80	8048
TEST 28	NOV	104	COMB	NOV	80	8048
TEST 29	NOV	104	COMB	NOV	80	8048
TEST 30	NOV	104	COMB	NOV	80	8048
TEST 31	NOV	104	COMB	NOV	80	8048
TEST 32	NOV	104	COMB	NOV	80	8048
TEST 33	NOV	104	COMB	NOV	80	8048
TEST 34	NOV	104	COMB	NOV	80	8048
TEST 35	NOV	104	COMB	NOV	80	8048
TEST 36	NOV	104	COMB	NOV	80	8048
TEST 37	NOV	104	COMB	NOV	80	8048
TEST 38	NOV	104	COMB	NOV	80	8048
TEST 39	NOV	104	COMB	NOV	80	8048
TEST 40	NOV	104	COMB	NOV	80	8048
TEST 41	NOV	104	COMB	NOV	80	8048
TEST 42	NOV	104	COMB	NOV	80	8048
TEST 43	NOV	104	COMB	NOV	80	8048
TEST 44	NOV	104	COMB	NOV	80	8048
TEST 45	NOV	104	COMB	NOV	80	8048
TEST 46	NOV	104	COMB	NOV	80	8048
TEST 47	NOV	104	COMB	NOV	80	8048
TEST 48	NOV	104	COMB	NOV	80	8048
TEST 49	NOV	104	COMB	NOV	80	8048
TEST 50	NOV	104	COMB	NOV	80	8048
TEST 51	NOV	104	COMB	NOV	80	8048
TEST 52	NOV	104	COMB	NOV	80	8048
TEST 53	NOV	104	COMB	NOV	80	8048
TEST 54	NOV	104	COMB	NOV	80	8048
TEST 55	NOV	104	COMB	NOV	80	8048
TEST 56	NOV	104	COMB	NOV	80	8048
TEST 57	NOV	104	COMB	NOV	80	8048
TEST 58	NOV	104	COMB	NOV	80	8048
TEST 59	NOV	104	COMB	NOV	80	8048
TEST 60	NOV	104	COMB	NOV	80	8048
TEST 61	NOV	104	COMB	NOV	80	8048
TEST 62	NOV	104	COMB	NOV	80	8048
TEST 63	NOV	104	COMB	NOV	80	8048
TEST 64	NOV	104	COMB	NOV	80	8048
TEST 65	NOV	104	COMB	NOV	80	8048
TEST 66	NOV	104	COMB	NOV	80	8048
TEST 67	NOV	104	COMB	NOV	80	8048
TEST 68	NOV	104	COMB	NOV	80	8048
TEST 69	NOV	104	COMB	NOV	80	8048
TEST 70	NOV	104	COMB	NOV	80	8048
TEST 71	NOV	104	COMB	NOV	80	8048
TEST 72	NOV	104	COMB	NOV	80	8048
TEST 73	NOV	104	COMB	NOV	80	8048
TEST 74	NOV	104	COMB	NOV	80	8048
TEST 75	NOV	104	COMB	NOV	80	8048
TEST 76	NOV	104	COMB	NOV	80	8048
TEST 77	NOV	104	COMB	NOV	80	8048
TEST 78	NOV	104	COMB	NOV	80	8048
TEST 79	NOV	104	COMB	NOV	80	8048
TEST 80	NOV	104	COMB	NOV	80	8048
TEST 81	NOV	104	COMB	NOV	80	8048
TEST 82	NOV	104	COMB	NOV	80	8048
TEST 83	NOV	104	COMB	NOV	80	8048
TEST 84	NOV	104	COMB	NOV	80	8048
TEST 85	NOV	104	COMB	NOV	80	8048
TEST 86	NOV	104	COMB	NOV	80	8048
TEST 87	NOV	104	COMB	NOV	80	8048
TEST 88	NOV	104	COMB	NOV	80	8048
TEST 89	NOV	104	COMB	NOV	80	8048
TEST 90	NOV	104	COMB	NOV	80	8048
TEST 91	NOV	104	COMB	NOV	80	8048
TEST 92	NOV	104	COMB	NOV	80	8048
TEST 93	NOV	104	COMB	NOV	80	8048
TEST 94	NOV	104	COMB	NOV	80	8048
TEST 95	NOV	104	COMB	NOV	80	8048
TEST 96	NOV	104	COMB	NOV	80	8048
TEST 97	NOV	104	COMB	NOV	80	8048
TEST 98	NOV	104	COMB	NOV	80	8048
TEST 99	NOV	104	COMB	NOV	80	8048
TEST 100	NOV	104	COMB	NOV	80	8048

August 1979

THE HSE-69 DEVELOPMENT TOOL

Need Emulators for Computers

Intel Microcomputer Generation

Micro

Application of microcomputer

terminated by commands from the keyboard. Execution may be suspended or display set "alive". Execution may be suspended or display set "alive".

A Hiccup

INTEL CORPORATION ASSUMES NO RESPONSIBILITY FOR THE USE OF ANY CIRCUITRY OTHER THAN CIRCUITRY EMBODIED IN AN INTEL PRODUCT. NO OTHER CIRCUIT PATENT LICENSES ARE IMPLIED.
©INTEL CORPORATION, 1979

I. PURPOSE AND SCOPE

This Application Note presents a description of the design and operation of a high-speed emulator for the Intel® MCS-48™ family of single chip microcomputers. The HSE-49™ emulator provides a simple and inexpensive means for executing and debugging 8049 programs which require the full 11-MHz operating speed of the part.

Section II of this Application Note describes some of the features of this development tool and how it may be used. Section III briefly discusses the hardware used to implement these features, while Section IV describes the manner in which program execution status is made available to the operator.

A detailed description of all of the operator commands is presented in Section V of this note, along with the modifiers and options which may be specified for each command. Known restrictions and limitations of the HSE-49 system are listed and explained in Section VI. Section VII shows how the basic circuit may be modified to provide options on memory organization, I/O configurations, etc.

Full schematics of the system hardware, as well as monitor software listings, are presented in Appendices A and B, respectively. A short summary of the command syntax is presented in Appendix C. Appendix D explains the error message codes which may appear during use.

It is assumed that the reader is already familiar with the operation of the 8048 or 8049 microcomputers. Some knowledge of the 8048 architecture is needed to understand sections of the command and modifier descriptions. Most users will already have this background. Other readers are referred to the *MCS-48 Microcomputer User's Manual*, Intel publication number 9800270.

II. THE HSE-49 DEVELOPMENT TOOL

In essence, the HSE-49 emulator provides the user a means for executing an MCS-48 program located in external RAM rather than internal ROM or EPROM. This allows programs being debugged to be modified easily and quickly during the debug cycle. A user's program may be entered into system RAM either manually or via a serial link from a host computer such as an Intellec® Microcomputer Development System. Once loaded, the program can be modified using an on-board keyboard and display, and executed in real-time in a number of breakpoint modes. The internal state of the processor, including RAM, accumulator, timer/counter, and status register contents, can also be read and modified through the keyboard.

Breakpoint and debug facilities are extremely flexible. The following execution modes are provided.

- Programs may be run in full (11 MHz) real time;
- Programs may be single-stepped;
- In break mode, programs run in full real time until break occurs;

- Breaks may be triggered by either program or external data RAM accesses;
- Any number of breakpoints may be used in any combination;
- "Auto-Step" operation causes the current program counter and Accumulator contents to be printed on the display for a short time on every instruction cycle;
- "Auto-Break" provides the above display only when a break flag is encountered, with real time operation otherwise;
- While running in non-break mode, a TTL-level pulse is generated whenever a break flag is encountered. This signal may be used to trigger an oscilloscope or Logic Analyzer to assist in hardware and software debug.
- While running in any mode, the keyboard and display are "alive". Execution may be suspended or terminated by commands from the keyboard.

Intent of this Note

While the HSE-49 emulator can assist a new microcomputer user in becoming familiar with the 8048 and 8049 microcomputers, its inherent debug capabilities will also prove helpful to design engineers. The design could be used for new system development and verification or adapted for prototype production.

The main concern in designing the HSE-49 emulator was to keep the basic design simple, while maximizing the system's flexibility. The design allows the use of jumpers, hardware and software switches, etc. to allow the user to reconfigure the system according to the way he dedicates chip-select pins, I/O, etc. The emulator can be changed to fit each user's unique needs, rather than forcing the user to alter his needs to what is provided.

The primary intent of note is to provide the reader with the information needed to reconstruct and make full use of the HSE-49 emulator. Less emphasis is placed on describing how the hardware operates or how the commands are implemented. This information may be found in the schematic diagrams and software listings included in the Appendices.

III. GENERAL HARDWARE OVERVIEW

User Program Emulation

The actual emulation of the user's program is done using an 8039 microcomputer (IC29 on the schematics in Appendix A) executing a program stored in external RAM. The basic minimum configuration includes the 8039 microcomputer, an 8282 address latch (IC19), and 2K bytes of 2114 RAM to use for program development and real-time execution (ICs B1, C1, B2, and C2). Additional RAM may be added to allow the user to expand his program and data memory to 4K each. (If an 11-MHz crystal is used with the microcomputer, type 2114-3 RAMs must be used.)

System Supervision

A second microcomputer — another 8039 (IC25) with an 8282 address latch (IC16) and off-chip program memory in a 2716 EPROM (IC15) — is used to scan the on-board keyboard and display, interpret and implement commands, drive serial interfaces, etc. In general, the master processor is used to interface the execution processor's memory spaces with the outside world and control the operation of the execution processor. In this note the two processors will be abbreviated "MP" and "EP", respectively. Figure 1 shows how the two processors interrelate with the rest of the system.

Keyboard/Display

The 33-key keyboard shown in Figure 2 includes a 16-key hexadecimal keypad and 17 special function keys for specifying commands and modifiers. Readers already

familiar with the PROMPT-48™ debug tool for the 8048 will find that 25 of the HSE-49 emulator keys are identical in function and layout to the PROMPT-48 keyboard, and use the PROMPT-48 command syntax. The eight additional keys are used to generalize and augment the PROMPT-48 capabilities, as described in Section V.

The eight-character seven-segment display (DS1-DS8) is used for displaying addresses, data, and pseudo-alphanumeric messages. The display responses printed in Section V and throughout this note use a mix of upper and lower case letters to indicate what seven-segment patterns appear. An 8243 (IC9) and eight DIP packages (resistor packs, current buffers, etc.) are used for multiplexing the display and scanning the keyboard.

Breakpoint Detection

Breakpoints are specified and detected using a 2102A 1K × 8 RAM corresponding to each pair of 2114s (ICs A1

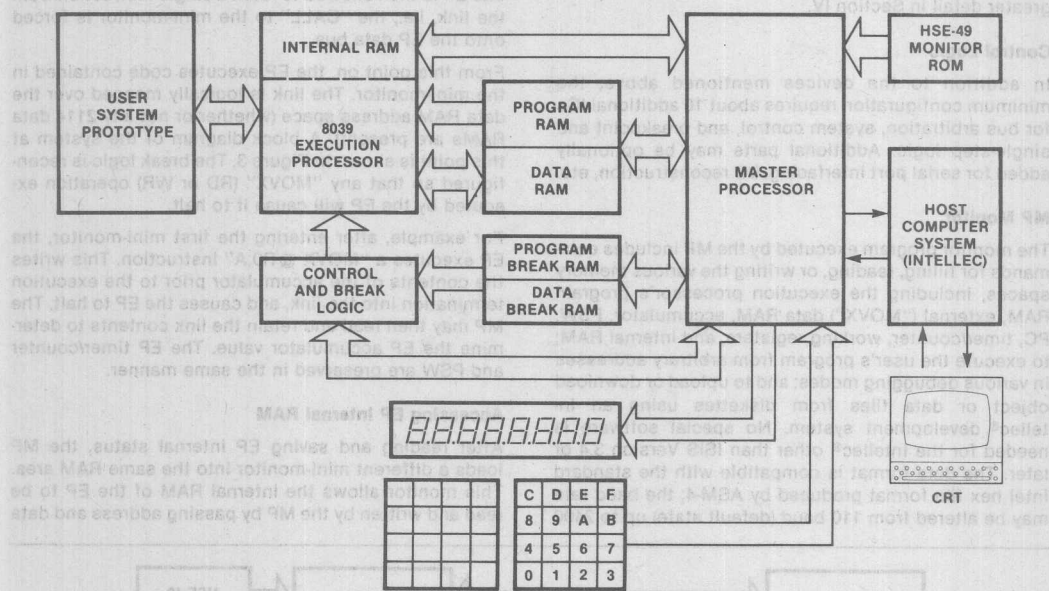


Figure 1. HSE-49™ Emulator Signal Flow Diagram

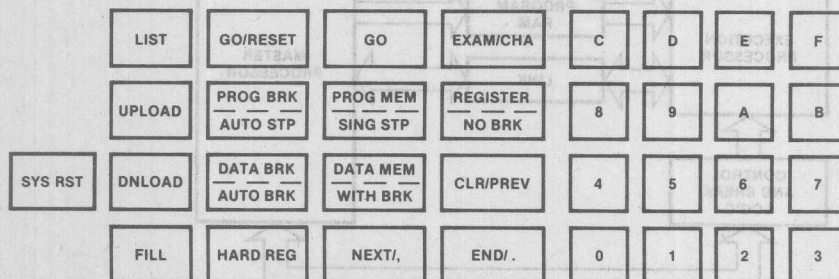


Figure 2. HSE-49™ Emulator Command Keyboard Organization

and A2). In effect, each program or data address accesses a 9-bit word. Eight bits are used normally for code or data storage. The ninth bit, accessed in parallel with the other eight, is used to indicate if a breakpoint has been set for that address. This output, when asserted, is latched (IC27 and IC36) and used to halt the execution processor via the single-step input. (In other modes, the break logic can be reconfigured to set the break requested flip-flop on any EP machine cycle or any EP "MOVX" instruction.)

Link Register

An 8212 8-bit latch (IC18) is used to communicate data and commands between the master and control processors. Under control of the MP, this register, called the "Link" register, may be logically mapped into either the program or data RAM address spaces. When this is done, the 2114s in the respective memory space are disabled and the link responds to all accesses, regardless of address. The link will be discussed in greater detail in Section IV.

Control Logic

In addition to the devices mentioned above, the minimum configuration requires about 10 additional ICs for bus arbitration, system control, and breakpoint and single-step logic. Additional parts may be optionally added for serial port interfacing, I/O reconstruction, etc.

MP Monitor

The monitor program executed by the MP includes commands for filling, reading, or writing the various memory spaces, including the execution processor's program RAM, external ("MOVX") data RAM, accumulator, PSW, PC, timer/counter, working registers, and internal RAM; to execute the user's program from arbitrary addresses in various debugging modes; and to upload or download object or data files from diskettes using an Intellec® development system. No special software is needed for the Intellec® other than ISIS Version 3.4 or later. The data format is compatible with the standard Intel hex file format produced by ASM-4; the baud rate may be altered from 110 baud (default state) up to 2400

baud from the on-board keyboard. Blocks of data may be transmitted to a CRT or printer and displayed in a tabular format.

IV. INTERPROCESSOR COMMUNICATION

Program Break Sequence

When the MP detects that the EP has been halted by the breakpoint hardware, or when the operator presses a key while the program is executing, the program break sequence is initiated. The low-order 23 bytes of user program memory is read into a buffer within the internal RAM of the MP. A short program for reading and transmitting internal EP status is written over the low-order program memory. (This is one of several "mini-monitors" overlaid over the user program area.) The link register is mapped logically over the user program memory, and loaded with the 8049 machine code for a "CALL" instruction to the mini-monitor program area. The EP is then allowed to fetch a single instruction from the link, i.e., the "CALL" to the mini-monitor is forced onto the EP data bus.

From this point on, the EP executes code contained in the mini-monitor. The link is logically mapped over the data RAM address space (whether or not any 2114 data RAMs are present). A block diagram of the system at this point is shown in Figure 3. The break logic is reconfigured so that any "MOVX" (RD or WR) operation executed by the EP will cause it to halt.

For example, after entering the first mini-monitor, the EP executes a "MOVX @R0,A" instruction. This writes the contents of the accumulator prior to the execution termination into the link, and causes the EP to halt. The MP may then read and retain the link contents to determine the EP accumulator value. The EP timer/counter and PSW are preserved in the same manner.

Accessing EP Internal RAM

After reading and saving EP internal status, the MP loads a different mini-monitor into the same RAM area. This monitor allows the internal RAM of the EP to be read and written by the MP by passing address and data

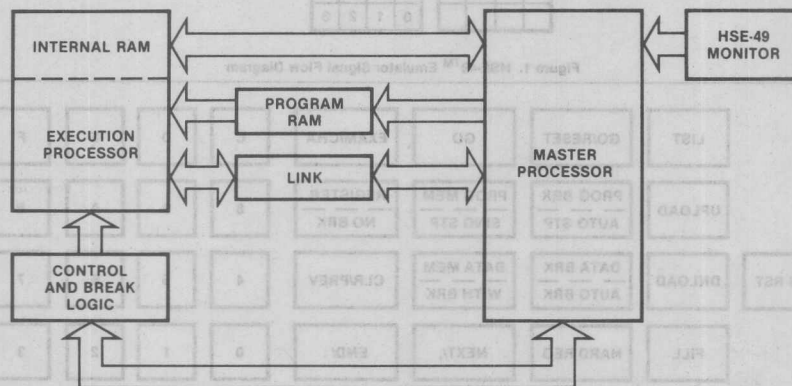


Figure 3. Communication between EP & MP

values between the two processors using the link register.

This is needed for two reasons. First, the EP program counter prior to the forced "CALL" instruction may be derived from the EP stack contents, and may be modified to cause the EP to resume execution at any desired address. Secondly, the internal RAM of the EP may then be accessed and modified in the process of executing a number of the monitor commands.

Resuming User Program Execution

In order to resume user program execution, a status-restoration mini-monitor is overlaid. This restores the EP internal status using a scheme analogous to the one in which the status was originally saved. The final step of the last mini-monitor is an "RETR" instruction, after which the EP is again halted. The low-order program memory saved earlier is rewritten into the appropriate area, the break logic is reconfigured for the desired execution mode, and the EP is released to run at full speed until the next break situation is encountered.

Note that all commands are implemented using "logical" rather than "physical" addressing. Thus the operator need not be concerned with the intricacies of the system design. For example, when any monitor command refers to low-order user program memory, the appropriate byte of storage within the MP internal RAM is accessed instead. If the location is altered, the internal RAM is modified appropriately. When program memory is reloaded prior to resuming user program execution, the modified version of the user program will be the one loaded.

Baud	HR06	HR07
110	93H	04H
150	96H	03H
300	45H	02H
600	9DH	01H
1200	44H	01H
2400	1AH	01H

Table 1. Serial Interface Data Rate Parameters

V. HSE-49 COMMAND DESCRIPTION

Whenever the characters "HSE-49" are present on the system display, a command string may be entered by the operator. In general, all command strings consist of a basic command initiator, an optional command modifier or type-designator, and a number of parameters or delimiters entered as hexadecimal digits. A command is executed, or a command in progress terminated, by pressing the [END/] key. Logical default values are assumed for the modifier and parameters if either (or both) are omitted. A default parameter assumed for the command modifier will be presented on the display when the first parameter is entered.

Each parameter is a string of up to three hexadecimal digits. If more than three digits are entered, only the most recent three are considered. This allows an erroneous digit to be corrected without respecifying the entire command. A parameter is completed by pressing the [NEXT/] key. Some commands may only need the

low order part of a parameter; i.e., a command incorporating a data byte (such as [FILL]) will use only the low-order 8 bits of the corresponding parameter; Internal RAM and hardware register addressing uses only seven. In each case, higher order bits are ignored.

A command string is terminated and the command invoked by pressing the [END/] key. The command will also be invoked by pressing the [NEXT/] key when no additional parameters are allowed. A command string may be aborted at any point before the command is invoked by pressing the [CLEAR/PREV] key, and the sign-on message will appear.

Errors

An illegal command string, command terminator, or hardware failure will cause an error message and error code number to appear on the display (e.g., "Error-3"). When this occurs, the monitor can be returned to command mode by pressing the [CLEAR] or [END/] keys. An explanation of the various error codes is given in Appendix D.

Command Classes

Commands for the HSE-49 emulator are divided into general classes, where all commands in each class have the same choice of options or modifiers. A brief description of each command, followed by a description of the allowed options, is presented below by class.

Data Manipulation/Control Command Group

Commands:

[EXAM/CHA]

Display Response — "Ech."

Function — Examine/change memory location.

Causes the memory address specified to be read and presented on the display. New data may be entered (if desired) from the hexadecimal keypad. New data is verified before appearing on the display. Subsequent or previous locations may be read by pressing the [NEXT/] or [PREV] keys, respectively. Command terminated with [END/] key.

[FILL]

Display Response — "FIL."

Function — Fill range of memory addresses with a single data value.

Fill the appropriate memory space between the addresses specified by the first two parameters with the low-order byte of the third parameter. If second parameter less than first, only the location specified by the first is affected. If third parameter omitted, zero is assumed. If second and third parameters omitted, individual address specified is cleared. Command is useful for setting a large range of breakpoints; e.g., all of page 3 may be enabled for break with the command:

[FILL][PROG BRK]<300>[,<3FF>[,<1>[.]

[LIST]

Display Response — "Lst."

Function — List memory to output device through HSE-49 serial port.

Display the contents of a range of addresses given by two parameters to a teletype or CRT screen. Data is formatted, 16 separated bytes per line, with the starting address of each line printed. If used with an Intellec® system, the operator first uses ISIS-II to transfer the TTY input to the CRT output ("COPY :TI: TO :CO:") then invokes this command from the keypad. Alternatively, any ISIS device or disk file name(:TO:, :LP:, :F1:HRDREG.SAV, etc.) may be used as the destination.

[DNLOAD]

Display Response — "dnL."

Function — Download memory through HSE-49 serial port

Load data in hex file format through the serial input port. If used with Intellec® system, the operator first invokes this command from the keypad, then uses ISIS-II to transfer a disk file to the teletype port ("COPY :Fn:file.HEX TO :TO:").

The use of the checksum field for the download command is expanded slightly over the Intel hex file format standard. If the first character of the checksum field is a question mark ("?"), the checksum for that record will not be verified. This allows large object files produced by the assembler to be patched using the ISIS text editor without the necessity of manually recomputing the checksum value.

[UPLOAD]

Display Response — "UPL"

Function — Upload memory through HSE-49 serial port.

Output the contents of a range of addresses specified by the two parameters through the HSE-49 serial port in standard Intel hex file format. If used with Intellec® system, the operator first uses ISIS-II to transfer the TTY input to a disk file ("COPY :TI: TO :Fn:file.HEX"), then invokes this command from the keypad.

Data types allowed:

[PROG MEM]

Display Response — "Pr."

Function — User program memory.

Memory used to develop and execute user program. Addresses 000 through 7FF are the execution processor's memory bank 0; 800 through FFF are memory bank 1.

[REGISTER]

Display Response — "rG."

Function — Register memory and RAM.

Internal RAM of execution processor. Locations 0-7 are working register bank 0; 18-1F are working register bank 1. Only the low-order 7 bits of an address are considered.

[DATA MEM]

Display Response — "dA."

Function — External data memory (if installed).

Memory accessed by execution processor "MOVX A,@Rr" or "MOVX @Rr,A" instructions. High-order 4 bits may or may not be relevant, depending on jumpering option selected (explained in Section VII of this note).

[HARD REG]

Display Response — "Hr."

Function — Hardware registers.

The execution processor (EP) hardware registers (accumulator, timer/counter, etc.), as well as several parameters for controlling HSE-49 system status, are accessible through this catch-all memory space. Addresses are as follows:

00 — EP accumulator.

01 — EP PSW.

Bits correspond to 8049 PSW except that bit 3 (unused in the 8049) is used to monitor and alter the state of F1. Bits 2-0 correspond to the stack pointer value after the EP executes a CALL to the mini-monitor; i.e., one greater than when EP was running the user's program.

02 — EP timer/counter.

03 — EP internal RAM location 00.

(This value is also accessible through [REGISTER] space.)

04 — EP program counter (low byte).

05 — EP program counter (high nibble).

06-07 — HSE-49 serial interface baud rate parameters. Defaults to 110 baud; other rates may be selected by loading the values listed in Table 1.

08 — HSE-49 automatic sequencing rate parameter. Used in [GO][AUTO STP] and [GO][AUTO BRK] execution commands. 00 → fastest; FF → slowest. Defaults to 20H; approximately two steps per second.

09 — Monitor version/release number (packed BCD).

0A-0F — Currently unused by the monitor program.

10-7F — Variables used by master processor (MP) monitor. Should not be altered by operator.

[PROG BRK]

Display Response — "Pb."

Function — User program breakpoint memory.

Memory space used to indicate points where program execution should halt when running in a mode with breakpoints enabled ([GO][W/ BRK] and [GO][AUTOBRK]). Break will occur if enabled byte is read as the first or last byte of a 2-byte instruction, or read in executing a MOV_P, MOV_P3, or JMPP instruction. Memory is only 1 bit per location; 00 indicates continue, 01 causes a halt. Addresses 000 through 7FF are the execution processor's memory bank 0; 800 through FFF are memory bank 1.

[DATA BRK]

Display Response — "db."

Function — External data RAM breakpoint memory.

Memory space used to indicate points where data accesses should halt when running in a mode with breakpoints enabled ([GO][W/ BRK] and [GO][AUTOBRK]). Memory is only 1 bit per location; 00 indicates continue, 01 causes a halt. High-order 4 bits of breakpoint address may or may not be relevant, dependent on jumpering option selected for the corresponding data RAM (explained in Section VII of this note).

User Program Execution Control Group

Commands:

[GO]

Display Response — "Go."

Function — Begin execution.

If a parameter is given as part of the command string, execution will begin at that address. Otherwise, the EP program counter (hardware registers 04 and 05) will be used. These will contain the program counter from an earlier program execution break unless they have since been explicitly modified by the operator.

If command is terminated by [END/], the EP's F1, PSW and stack pointer will be cleared. If command string is terminated by [NEXT/], PSW will be taken from the EP PSW contents (hardware register 01).

While running the user's program, the characters "-run-" are written on the display. Execution may be halted and another command initiated by pressing the appropriate command key. Execution may be suspended at any time in any mode by pressing the [END/] key. This will cause the current value of the execution processor program counter and accumulator to appear on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. At this point, or when an enabled breakpoint is encountered, pressing the [NEXT/] key will cause the program to continue in the same mode as before. Any other command may be invoked by pressing the appropriate command string.

[GO/RESET]

Display Response — "Gr."

All mnemonics copyrighted © Intel Corporation 1976.

Function — Go from reset state.

EP is hardware-reset and released to execute the user's program from location 000H. No parameters are allowed. F0, F1, PSW, stack pointer, memory bank flip-flop, etc., are cleared.

Note that this command does not require the use of mini-monitors to initiate program execution. As the last phase of the program development cycle, the 2114 program RAMs and address decoder may be removed and replaced by a ROM or EPROM part (not shown in schematics). This command may be used to start execution when the program RAM has been removed. No interrogation of EP status or internal RAM may be done, nor are break or single-step modes allowed in this case, though the 2102A breakpoint RAM outputs may still be used to trigger a logic analyzer.

Execution modes allowed:

[NO BRK]

Display Response — "nb."

Function — Without breakpoints.

Full-speed execution without breakpoints enabled. Does not affect the state of the breakpoint memories.

[SING STP]

Display Response — "SSt."

Function — Single Step.

Step through program one instruction at a time. After each instruction is executed, execution halts with the current value of the Execution Processor Program Counter and Accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate Hardware Registers. At the point, [NEXT/] will cause the program to execute one more instruction, or any other command may be invoked by pressing the appropriate command string. Does not affect the state of the Breakpoint Memories.

[W/ BRK]

Display Response — "br."

Function — With breakpoints.

Full-speed execution with breakpoints enabled. When a breakpoint is encountered, execution halts with the current value of the execution processor program counter and accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. At this point, [NEXT/] will cause the program to continue until the next breakpoint is reached, or any other command may be invoked by pressing the appropriate command string.

[AUTO STP]

Display Response — "ASt."

Function — Automatically sequence through a series of instructions.

Step through program one instruction at a time. After each instruction is executed, execution halts with the current value of the execution processor program counter and accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. Execution resumes after a time determined by contents of hardware register 08. Does not affect the state of the breakpoint memories.

[AUTO BRK]

Display Response — "Abr."

Function — Automatically sequence between breakpoints.

Execute a series of instructions in real time between breakpoints. When breakpoint is encountered, halt EP temporarily while program counter and accumulator contents are displayed, then continue. Display is sustained after execution resumes. Does not affect the state of the breakpoint memories.

Breakpoint Control Command Group

Commands:

[B]

Display Response — "Stb."

Function — Breakpoint set.

Set breakpoint for the address given. Multiple breakpoints may be set by entering additional addresses, separated by the [NEXT/] key. Command terminated by pressing [END/]. Action taken is to fill the appropriate breakpoint memory locations with logical ones.

[C]

Display Response — "CLb."

Function — Clear breakpoint.

Clear breakpoint for the address given. Multiple breakpoints may be cleared by entering additional addresses, separated by the [NEXT/] key. Command terminated by pressing [END/]. Action taken is to fill the appropriate breakpoint memory locations with logical zeroes.

Data types allowed:

[PROG MEM]

Display Response — "Pr."

Function — Break on program memory fetch.

Applies command to the program breakpoint memory space.

[DATA MEM]

Display Response — "dA."

Function — Break on data memory access.

Applies command to the external data breakpoint memory space.

System Control Command Group

Command:

[SYS RST]

Display Response — "HSE-49."

Function — System reset.

Reset both the MP and EP and clear all breakpoints (requires approximately one second). CAUTION — If reset while EP is executing the user's program, the low order section of program memory (about 23 bytes) will be altered.

VI. SYSTEM LIMITATIONS

In designing the HSE-49 emulator, certain compromises were made in an attempt to maximize the usefulness of the emulator while keeping the circuitry simple and inexpensive. As a result, the following limitations exist and must be taken into account when using the system.

1. As explained in Section IV, user program execution is terminated (by single-stepping, breakpoints, pressing the [END/] key, etc.) by forcing the execution processor to execute a "CALL" instruction to the mini-monitor. This uses one level of the EP subroutine stack. The EP PSW reflects the value of the stack pointer *after* processing this CALL. As a result, the value indicated for stack depth by examining the EP PSW (hardware register 01) is one greater than the depth when the break was initiated. The user program must not be using all eight levels of stack when a break is initiated or the bottom level will be destroyed.
2. User program is initiated (by the [GO] command or when resuming execution after a breakpoint, single-stepping, etc.) by forcing the EP to execute an "RETR" instruction. This will clear the EP interrupt-in-progress flip-flop. If the user program allows both external and timer interrupts to be enabled at the same time, care must be taken to avoid causing a break while the EP is within an interrupt servicing routine. No limitation is placed on breakpoints or single-stepping in the background program because of this.
3. When the user program execution is terminated (by a break, single-stepping, etc.) and later resumed, the EP timer/counter is restored to its value when the break occurred (unless modified by the user). The prescaler, however, will have changed. Thus, up to 31 machine cycles may be "lost" or "gained" if a break occurs while the timer is running.
4. Timer interrupts occurring at the same time as an EP break may be ignored if the timer overflow occurs after breaking user program execution before the timer value is saved.
5. The 8049 "RET" and "RETR" instructions are each 1-byte, 2-cycle instructions. During the second cycle the byte following the return instruction is fetched and ignored. If a program breakpoint is set for a location following a "RET" or "RETR" instruction, a break will be initiated when the return is executed.

6. Breakpoints should not be placed in the last 3 bytes of an EP memory bank (locations 7FDH-7FFH and 0FFDH-0FFFH). User program should not be single-stepped or auto-stepped through these locations.
7. Since I/O configuration is determined by external hardware rather than software, I/O modes may not be altered while a program is executing. (See Section VII for further details.)
8. The "ANL BUS,#nn" and "ORL BUS,#nn" instructions may not be used in the user program, as external hardware cannot properly restore these functions.
9. The memory bank select flag is not affected by the user program break sequence. Upon resuming execution with the [GO] command this flag will remain in the same state as before the preceding break. The flag may be cleared only by executing the [GO/RESET] or [SYS RST] commands.

VII. HARDWARE CONFIGURATIONS

A number of control and status lines are available to the user. All are low-power Schottky TTL-compatible signals.

TP1 — Unused MP input.

TP2 — Unused MP output.

TP3 — User program suspended. Low when EP running user code. High when halted or running mini-monitors.

TP4 — Breakpoint encountered. Normally low. High-level pulse generated when breakpoint passed. Useful for triggering logic analyzers, oscilloscopes, etc.

TP5 & TP6 — Memory matrix mode control. Select program vs. data RAM, link mapping configuration, etc. (See Appendix B for details.)

TP7 — Bus control. Low when MP controls common memory buses. High when EP controls memory buses.

The HSE-49 emulator hardware is designed to allow the user to reconfigure the system for a wide variety of different applications by installing or removing jumper wires or additional components. The schematics in Appendix A show the components needed for a variety of different configurations. In general, not all of the devices are required (or allowed) for any one configuration. The devices which are required are included in the following description.

The types of options allowed are divided below into several general classes and subdivided into mutually-independent features. Within some of these features there are numbered, mutually exclusive configurations; i.e., the serial interface (if desired) may use either

current-loop or RS-232C current buffers, but not both at one time.

Standard Operating Configuration

(Minimum system configurations — up to 4K program RAM; no data RAM; no serial interfaces; no execution processor I/O reconstruction.)

A. Basic 2K monitor from Appendix B:

- Install resistors R4-R6
- Install transistor Q1
- Install crystals Y1-Y2
- Install capacitors C5-C38
- Install switches S1-S33
- Install displays DS1-DS8
- Install IC1-IC2
- Install RP3-RP5
- Install IC6-IC7
- Install RP8
- Install IC9
- Install IC15-IC20
- Install IC25-IC30
- Install IC34
- Install IC36-IC38
- Install A1-A2
- Install B1-B2
- Install C1-C3
- Install jumpers 13-15
- Install jumpers 17-18
- Install jumper 20

B. Expansion 2K monitor:

- Install IC14
- Remove jumper 17

Serial Interface Buffer Selection

A. Current loop serial interfaces (4N46s) installed for use with full Inteltec® Model 800 development system TTY port.

- Install IC21-IC22
- Install resistor R1-R3
- Install jumpers 4-9
- (Remove RS-232 jumpers)

B. RS-232C serial interfaces (MC1488 and MC1489) installed for use with CRT as output device for data dumps:

- Install IC23-IC24
- Install jumpers 1-3
- Install jumpers 10-11
- (Remove current-loop jumpers)

External Data RAM Address Decoding Scheme for Execution Processor

A. Up to 16 pages of on-board external data RAM installed for execution processor (addresses 0 through

0FFFH = 4K bytes); port 2 used for addressing pages 0 through 15:

- Install jumpers 21-25
- Install jumper 27
- Install A5-A8
- Install B5-B8
- Install C5-C8

B. One page of on-board external data RAM installed for execution processor (addresses 0 through 0FFFH); port 2 not used for data addressing:

- Install jumper 26
- Install jumper 28
- Install A5
- Install B5
- Install C5

Connect the outputs of IC20, pins 7, 9, 10, & 11 to the inputs of a 74LS21 AND gate (not shown). Connect the output to CE and CS inputs of A5-C5. (Note: these signals are all present at jumpers 21-24 on the schematics.)

Reconstructing I/O for Execution Processor

A. Application of port 2, pins P23-P20:

- (1) Using P23-P20 for latched output data (used with "OUTL P2,A", "ANL P2,#data", and "ORL P2,#data" instructions);

Install IC31

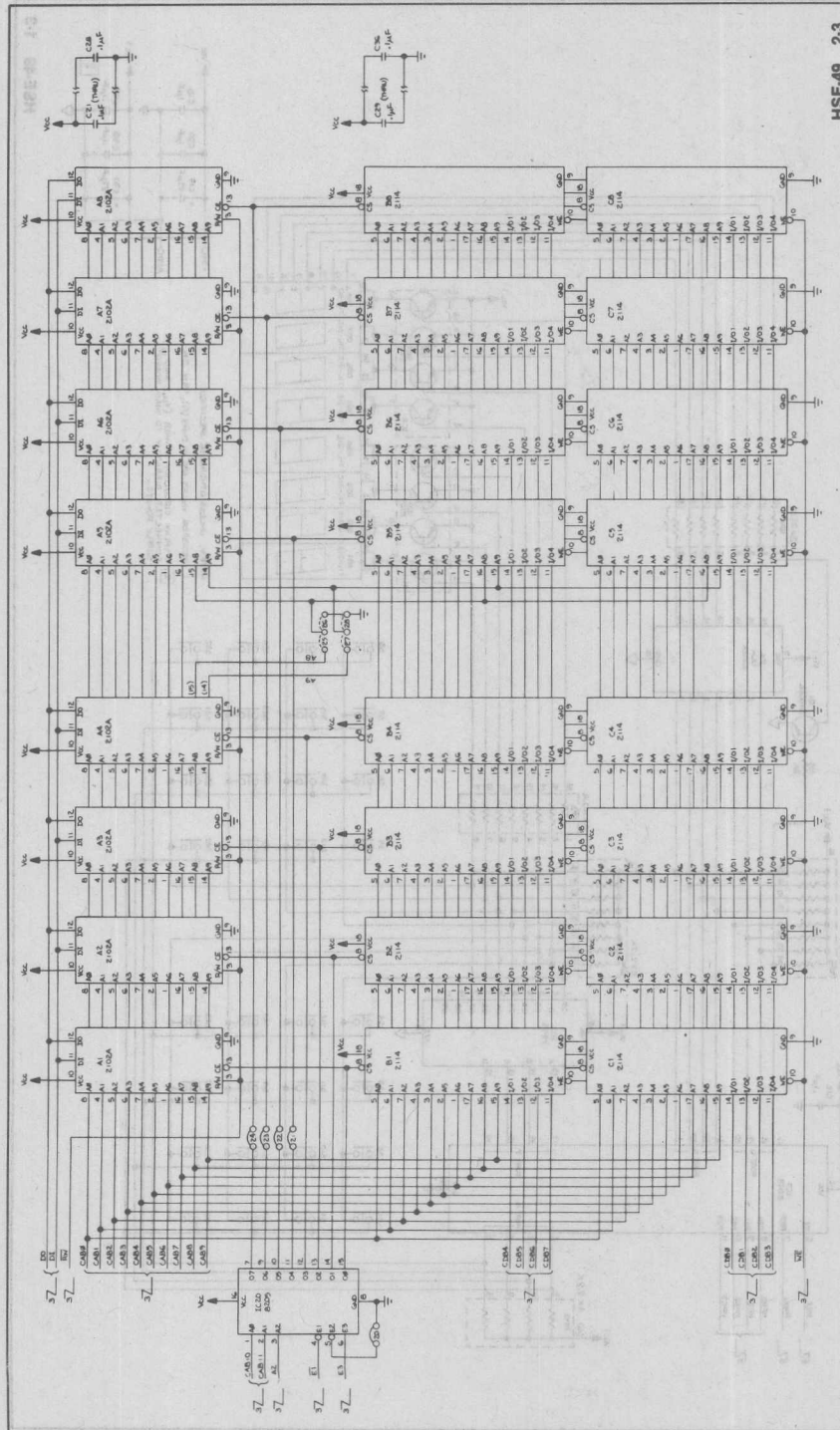
- (2) Using P23-P20 for interfacing to an 8243 in user's prototype:

Connect D3-D0 pins on IC31 socket to corresponding Q3-Q0 pins.

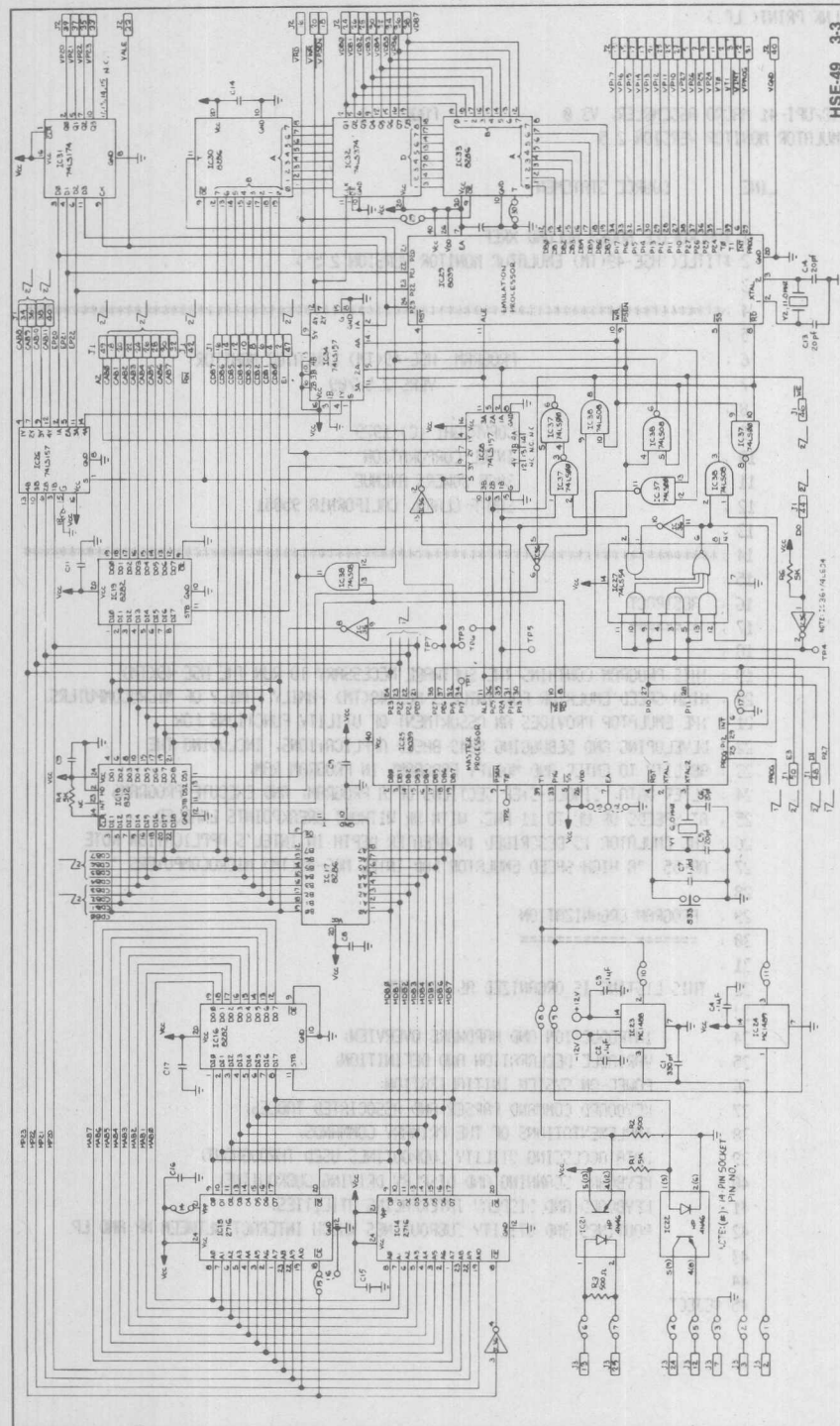
B. Application of execution processor BUS:

- (1) Use of BUS as latched output port ("OUTL BUS,A");

Install IC32



Ram Memory



HSE-49 3-3

Central Processor

ASM48 HSE49 LNK PRINT(LF.)

1515-11 MCS-48/UPI-41 MACRO ASSEMBLER, V3.0
HSE-49(TM) EMULATOR MONITOR VERSION 2.5

PAGE 1

```

LOC  OBJ      LINE      SOURCE STATEMENT
1  $MACROFILE NOGEN NOCOND XREF
2  $TITLE('HSE-49(TM) EMULATOR MONITOR VERSION 2.5')
3
4  ;*****
5
6  ;                                PROGRAM: HSE-49(TM) EMULATOR MONITOR
7  ;                                VERS 2.5/789
8
9  ;                                COPYRIGHT (C) 1979
10 ;                                INTEL CORPORATION
11 ;                                2065 BOWERS AVENUE
12 ;                                SANTA CLARA, CALIFORNIA 95051
13
14 ;*****
15
16 ;  ABSTRACT
17 ;  =====
18
19 ;  THIS PROGRAM CONTAINS THE SOFTWARE NECESSARY TO RUN THE HSE-49(TM)
20 ;  HIGH-SPEED EMULATOR FOR INTEL'S MCS-48(TM) FAMILY FAMILY OF MICROCOMPUTERS.
21 ;  THE EMULATOR PROVIDES AN ASSORTMENT OF UTILITY FUNCTIONS FOR
22 ;  DEVELOPING AND DEBUGGING 8048-BASED APPLICATIONS, INCLUDING THE
23 ;  ABILITY TO ENTER AND MODIFY PROGRAMS IN PROGRAM RAM,
24 ;  ALTER DATA, SINGLE-STEP SECTIONS OF A PROGRAM, AND EXECUTE PROGRAMS
25 ;  AT SPEEDS OF UP TO 11 MHz, WITH OR WITHOUT BREAKPOINTS ENABLED.
26 ;  THE EMULATOR IS DESCRIBED IN GREATER DEPTH IN INTEL'S APPLICATION NOTE
27 ;  AP-55 "A HIGH-SPEED EMULATOR FOR INTEL MCS-48(TM) MICROCOMPUTERS."
28
29 ;  PROGRAM ORGANIZATION
30 ;  =====
31
32 ;  THIS LISTING IS ORGANIZED AS FOLLOWS:
33
34 ;      INTRODUCTION AND HARDWARE OVERVIEW;
35 ;      VARIABLE DECLARATION AND DEFINITION;
36 ;      POWER-ON SYSTEM INITIALIZATION;
37 ;      KEYBOARD COMMAND PARSER AND ASSOCIATED TABLES;
38 ;      IMPLEMENTATIONS OF THE PRIMARY COMMANDS;
39 ;      DATA ACCESSING UTILITY SUBROUTINES USED THROUGHOUT;
40 ;      KEYBOARD SCANNING AND DISPLAY DRIVING SUBROUTINE;
41 ;      KEYBOARD AND DISPLAY INTERFACING UTILITIES;
42 ;      ROUTINES AND UTILITY SUBROUTINES WHICH INTERACT BETWEEN MP AND EP.
43
44
45 $EJECT

```


LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOLS	VALUES	LOC	OBJ
46		:					
47		:	INTRODUCTION AND HARDWARE OVERVIEW				
48		:	=====				
49		:					
50		:	THE EMULATOR DESIGN USES TWO MICROPROCESSORS. ONE PROCESSOR CONTROLS				
51		:	SYSTEM STATUS, INTERPRETS MONITOR COMMANDS, AND COMMUNICATES				
52		:	WITH THE OUTSIDE WORLD THROUGH THE ON-BOARD KEYBOARD, DISPLAY, SERIAL				
53		:	INTERFACES, CONTROL SIGNALS, ETC.				
54		:	A SECOND PROCESSOR IS USED TO ACTUALLY				
55		:	EXECUTE THE USER'S PROGRAM UNDER THE CONTROL OF THE FIRST.				
56		:	THESE PROCESSORS ARE REFERRED TO				
57		:	THROUGHOUT THIS PROGRAM AS THE MASTER PROCESSOR (MP) AND EXECUTION				
58		:	PROCESSOR (EP) RESPECTIVELY.				
59		:					
60		:	THE PROGRAM IN THIS LISTING IS EXECUTED BY THE MASTER PROCESSOR.				
61		:	AT THE END OF THIS LISTING ARE SEVERAL SHORT "MINI-MONITOR OVERLAYS"				
62		:	WHICH THE EXECUTION PROCESSOR EXECUTES WHEN INTERACTION BETWEEN THE				
63		:	TWO PROCESSORS IS NECESSARY.				
64		:					
65		:	THIS PROGRAM WAS WRITTEN USING A NUMBER OF MACROS TO HANDLE THE ALLOCATION				
66		:	OF MPU RESOURCES (WORKING REGISTERS, INTERNAL RAM, AND MP MONITOR ROM				
67		:	FOR CODE AND DATA STORAGE). THESE MACRO DEFINITIONS ARE INCLUDED IN A FILE				
68		:	NAMED "ALLOC.MAC" AND ARE PRINTED IN THIS LISTING FOR REFERENCE.				
69		:	ANOTHER SET OF MACROS IS USED TO SIMPLIFY THE ACCESSING OF VARIABLES				
70		:	STORED IN INTERNAL RAM (AS OPPOSED TO WORKING REGISTERS) BY USING R1 TO				
71		:	INDIRECTLY ADDRESS THE APPROPRIATE RAM LOCATION WHEN NECESSARY.				
72		:	THESE MACROS ARE INCLUDED IN "MOPCOD.MAC", AND ARE ALSO PRINTED HERE.				
73		:	COMPLETE UNDERSTANDING OF THESE MACROS IS NOT REQUIRED TO UNDERSTAND THE				
74		:	MONITOR PROGRAM. ALL LINES WHICH ACTUALLY PRODUCE OBJECT CODE APPEAR IN				
75		:	THE LISTING ITSELF, INDENTED TWO SPACES FROM THE NORMAL TABULATION COLUMNS.				
76		:	THE ACTUAL MONITOR PROGRAM FOR THE EMULATOR BEGINS AT APPROXIMATELY				
77		:	SOURCE LINE NUMBER 500.				
78		:					
79		:	LINES GENERATED BY MACRO EXPANSION ARE FLAGGED BY A PLUS SIGN ("+")				
80		:	IMMEDIATELY FOLLOWING THE SOURCE LINE NUMBER.				
81		:	A NUMBER OF LINES FROM THE VARIOUS MACRO DEFINITIONS WHICH DO NOT				
82		:	PRODUCE ANY OBJECT CODE ARE PROCESSED BY THE ASSEMBLER				
83		:	AS THESE MACROS ARE EXPANDED. WHEN THIS IS THE CASE, THESE LINES ARE				
84		:	SUPPRESSED FROM THE LIST FILE. AS A RESULT, THE LINE NUMBERS ARE				
85		:	NOT ALWAYS CONSECUTIVE WHERE A MACRO IS BEING INVOKED.				
86		:					
87		:	NOTE:				
88		:	=====				
89		:	"SOURCE-LINE" REFERS TO THE DECIMAL NUMBERS LEFT OF EACH INSTRUCTION.				
90		:	AT THE END OF THE LISTING IS AN ASSEMBLY CROSS-REFERENCE TABLE INDICATING				
91		:	THE SEQUENTIAL SOURCE-LINE NUMBER OF ALL INSTANCES WHERE ANY VARIABLE				
92		:	IS DEFINED OR REFERENCED. THIS WILL BE OF GREAT ASSISTANCE IN				
93		:	LOCATING SPECIFIC SUBROUTINES, ETC. IN THE LISTING.				
94		:					
95		:	MINEMONICS COPYRIGHT (C) 1976 INTEL CORPORATION				
96		:					
97		:	\$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	LINE	LOC	OBJ
		98	\$ INCLUDE(:F0,ALLUC,MAC)	98		
0000		= 99	?P1 SET 0	99		
		= 100	:	100		
0000		= 101	?R0 EQU 0	101		
0001		= 102	?R1 EQU 1	102		
0002		= 103	?R2 EQU 2	103		
0003		= 104	?CONST EQU 3	104		
0004		= 105	?R EQU 4 ; ACCUMULATOR VARIABLE TYPE	105		
		= 106	:	106		
		= 107	, THE FOLLOWING INITIALIZES THE LINKED LIST POINTERS FOR	107		
		= 108	, THE REGISTER ALLOCATION AND DEALLOCATION ROUTINES.	108		
		= 109	:	109		
0003		= 110	?B0R2 SET 3	110		
0004		= 111	?B0R3 SET 4	111		
0005		= 112	?B0R4 SET 5	112		
0006		= 113	?B0R5 SET 6	113		
0007		= 114	?B0R6 SET 7	114		
0008		= 115	?B0R7 SET 8	115		
		= 116	:	116		
0002		= 117	?B0PNT SET 2	117		
		= 118	:	118		
0003		= 119	?B1R2 SET 3	119		
0004		= 120	?B1R3 SET 4	120		
0005		= 121	?B1R4 SET 5	121		
0006		= 122	?B1R5 SET 6	122		
0007		= 123	?B1R6 SET 7	123		
0008		= 124	?B1R7 SET 8	124		
		= 125	:	125		
0002		= 126	?B1PNT SET 2	126		
		= 127	:	127		
0000		= 128	ORGPG0 SET 000H	128		
0100		= 129	ORGPG1 SET 100H	129		
0200		= 130	ORGPG2 SET 200H	130		
0300		= 131	ORGPG3 SET 300H	131		
0400		= 132	ORGPG4 SET 400H	132		
0500		= 133	ORGPG5 SET 500H	133		
0600		= 134	ORGPG6 SET 600H	134		
0700		= 135	ORGPG7 SET 700H	135		
		= 136	:	136		
		= 137	#EJECT	137		

```

= 138 ; *****
= 139 ;
= 140 ; START OF ALLOCATION MACROS
= 141 ;
= 142 ; *****
= 143 ;
= 144 ?MSAVE MACRO SYMBOL, BANK, PNTVAL
= 145 IF PNTVAL EQ 0
= 146 ERROR 2
= 147 EXITM
= 148 ENDIF
= 149 $ SAVE GEN
= 150 SYMBOL SET R&PNTVAL
= 151 $ RESTORE
= 152 ?BANK&PNT SET ?BANK&R&PNTVAL
= 153 ENDM
= 154 ;
= 155 ;
= 156 ?MINDX SET 20H
= 157 ;
= 158 ?MSAVE MACRO SYMBOL, LENGTH, ADDR
= 159 $ SAVE GEN
= 160 SYMBOL EQU ADDR
= 161 $ RESTORE
= 162 ?MINDX SET ?MINDX+LENGTH
= 163 ENDM
= 164 ;
= 165 MBLOCK MACRO SYMBOL, LENGTH
= 166 ?MSYMBOL EQU 3
= 167 ?MSAVE SYMBOL, LENGTH, ??MINDX
= 168 ENDM
= 169 ;
= 170 DECLARE MACRO SYMBOL, TYPE
= 171 ?MSYMBOL SET ?TYPE
= 172 IF ?TYPE EQ 2
= 173 ?MSAVE SYMBOL, 1, ??MINDX
= 174 EXITM
= 175 ENDIF
= 176 IF ?TYPE EQ 0
= 177 ?MSAVE SYMBOL, 0, ??B0PNT
= 178 EXITM
= 179 ENDIF
= 180 IF ?TYPE EQ 1
= 181 ?MSAVE SYMBOL, 1, ??B1PNT
= 182 EXITM
= 183 ENDIF
= 184 ENDM
= 185 ;
= 186 $ EJECT

```

0020

1

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LINE
= 187			*****		= 187
= 188			REORG MACRO TO RESET THE INSTRUCTION LOCATION COUNTER		= 188
= 189			TO THE FIRST FREE LOCATION ON THE FIRST PAGE MODULE WILL		= 189
= 190			FIT WITHIN		= 190
= 191			REORG MACRO LOCATION		= 191
= 192			\$SAVE GEN		= 192
= 193			ORG LOCATION		= 193
= 194			\$RESTORE		= 194
= 195			ENDM		= 195
= 196					= 196
= 197			CODEBLK MACRO TO FIND A PAGE OF ROM		= 197
= 198			WHICH THIS BLOCK OF CODE WILL FIT WITHIN		= 198
= 199			CODEBLK MACRO LENGTH		= 199
= 200			LENGTH SET LENGTH		= 200
= 201			IF HIGH(ORGP0+LENGTH-1) EQ 0		= 201
= 202			REORG %ORGP0		= 202
= 203			%START SET \$		= 203
= 204			EXITH		= 204
= 205			ENDIF		= 205
= 206			IF HIGH(ORGP1+LENGTH-1) EQ 1		= 206
= 207			REORG %ORGP1		= 207
= 208			%START SET \$		= 208
= 209			EXITH		= 209
= 210			ENDIF		= 210
= 211			IF HIGH(ORGP2+LENGTH-1) EQ 2		= 211
= 212			REORG %ORGP2		= 212
= 213			%START SET \$		= 213
= 214			EXITH		= 214
= 215			ENDIF		= 215
= 216			IF HIGH(ORGP4+LENGTH-1) EQ 4		= 216
= 217			REORG %ORGP4		= 217
= 218			%START SET \$		= 218
= 219			EXITH		= 219
= 220			ENDIF		= 220
= 221			IF HIGH(ORGP5+LENGTH-1) EQ 5		= 221
= 222			REORG %ORGP5		= 222
= 223			%START SET \$		= 223
= 224			EXITH		= 224
= 225			ENDIF		= 225
= 226			IF HIGH(ORGP6+LENGTH-1) EQ 6		= 226
= 227			REORG %ORGP6		= 227
= 228			%START SET \$		= 228
= 229			EXITH		= 229
= 230			ENDIF		= 230
= 231			IF HIGH(ORGP7+LENGTH-1) EQ 7		= 231
= 232			REORG %ORGP7		= 232
= 233			%START SET \$		= 233
= 234			EXITH		= 234
= 235			ENDIF		= 235
= 236			IF HIGH(ORGP3+LENGTH-1) EQ 3		= 236
= 237			REORG %ORGP3		= 237
= 238			%START SET \$		= 238
= 239			EXITH		= 239
= 240			ENDIF		= 240
= 241			ERROR 0 ;*** INSUFFICIENT SPACE FOR CODE ON ANY PAGE ***		= 241

LOC	OBJ	LINE	SOURCE STATEMENT	THOMAS/2 330002	TIME	LOC	OBJ
		= 242	ENDM		005		
		= 243	DATABLK INSERTS ONTO PAGE 3		005		
		= 244	DATABLK MACRO LENGTH		005		
		= 245	?LENGTH SET LENGTH		005		
		= 246	IF HIGH(ORGP3+LENGTH-1) EQ 3		005		
		= 247	REORG ?ORGP3		005		
		= 248	?START SET \$		005		
		= 249	EXITM		005		
		= 250	ENDIF		005		
		= 251	ERROR 0 ;*** INSUFFICIENT SPACE FOR DATA BLOCK ON PAGE 3 ***		005		
		= 252	ENDM		005		
		= 253	?SIZE PRINTS A LINE TO THE SOURCE FILE GIVING BLOCK SIZE,		005		
		= 254	AND UPDATES APPROPRIATE ORGP3		005		
		= 255	?SIZE MACRO BLK,PGE		005		
		= 256	\$SAVE GEN		005		
		= 257	SIZE SET BLK		005		
		= 258	;		005		
		= 259	*****		005		
		= 260	IF ?LENGTH LT SIZE		005		
		= 261	ERROR 0 ;*** SIZE EXCEEDS SPACE CHECKED FOR BY CODEBLK MACRO		005		
		= 262	ENDIF		005		
		= 263	IF HIGH(\$-1) NE HIGH(?START)		005		
		= 264	ERROR 0 ;*** CODE OR DATA BLOCK ROLLED OVER PAGE BOUNDARY ***		005		
		= 265	ENDIF		005		
		= 266	\$RESTORE		005		
		= 267	ORGP3&PGE SET \$		005		
		= 268	ENDM		005		
		= 269	?SIZECHK CHECKS SIZE OF PRECEDING BLOCK, PRINTS SIZE TO .LSI FILE		005		
		= 270	SIZECHK MACRO		005		
		= 271	?SIZE %(\$-?START),%HIGH(?START)		005		
		= 272	ENDM		005		
		= 273	;		005		
		= 274	;		005		
		= 275	?SOURCE CODE SPACE ALLOCATION SUMMARY STATEMENT		005		
		= 276	?SOURCE MACRO		005		
		= 277	\$SAVE LIST GEN		005		
		= 278	P6SIZE SET ORGP0-000H ;BYTES USED ON PAGE 0		005		
		= 279	P6SIZE SET ORGP1-100H ;BYTES USED ON PAGE 1		005		
		= 280	P6SIZE SET ORGP2-200H ;BYTES USED ON PAGE 2		005		
		= 281	P6SIZE SET ORGP3-300H ;BYTES USED ON PAGE 3		005		
		= 282	P6SIZE SET ORGP4-400H ;BYTES USED ON PAGE 4		005		
		= 283	P6SIZE SET ORGP5-500H ;BYTES USED ON PAGE 5		005		
		= 284	P6SIZE SET ORGP6-600H ;BYTES USED ON PAGE 6		005		
		= 285	P6SIZE SET ORGP7-700H ;BYTES USED ON PAGE 7		005		
		= 286	\$EJECT		005		
		= 287	\$RESTORE		005		
		= 288	ENDM		005		
		= 289	\$EJECT		005		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDSOURCE STATEMENT	FILE	LOC	OBJ
		290 ;					
		291 \$	INCLUDE(:F0:MOPCOD.MAC)				
		= 292 ;					
		= 293 ; ?FORM1 MACRO	FOR GENERALIZING OPCODE INSTRUCTION				
		= 294 ;					
		= 295 ?FORM1 MACRO	OPCODE, SRC				
		= 296 IF	?&SRC EQ 2				
		= 297 \$	SAVE GEN				
		= 298	MOV R1, #SRC				
		= 299	OPCODE				
		= 300 \$	RESTORE				
		= 301	EXITM				
		= 302 ENDIF					
		= 303 IF	?&SRC EQ 0 OR ?&SRC EQ 1				
		= 304 \$	SAVE GEN				
		= 305	OPCODE A, SRC				
		= 306 \$	RESTORE				
		= 307	EXITM				
		= 308 ENDIF					
		= 309 IF	?&SRC EQ 3				
		= 310 \$	SAVE GEN				
		= 311	OPCODE A, #SRC				
		= 312 \$	RESTORE				
		= 313	EXITM				
		= 314 ENDIF					
		= 315	ERROR 1				
		= 316 ENDM					
		= 317 ;					
		= 318 ; ?FORM2 MACRO	FOR GENERALIZING MOVES FROM THE ACC TO A VARIABLE				
		= 319 ?FORM2 MACRO	DEST				
		= 320 IF	?&DEST EQ 2				
		= 321 \$	SAVE GEN				
		= 322	MOV R1, #DEST				
		= 323	MOV @R1, A				
		= 324 \$	RESTORE				
		= 325	EXITM				
		= 326 ENDIF					
		= 327 IF	?&DEST EQ 0 OR ?&DEST EQ 1				
		= 328 \$	SAVE GEN				
		= 329	MOV @DEST, A				
		= 330 \$	RESTORE				
		= 331	EXITM				
		= 332 ENDIF					
		= 333	ERROR 1				
		= 334 ENDM					
		= 335 ;					
		= 336 ; ?FORM3 MACRO	FOR GENERALIZING MOVES FROM THE ACC TO A VARIABLE				
		= 337 ;	WHEN IT IS KNOWN THAT R1 (IF NEEDED FOR INDIRECT ADDRESSING)				
		= 338 ;	IS ALREADY PKESET.				
		= 339 ?FORM3 MACRO	DEST				
		= 340 IF	?&DEST EQ 2				
		= 341 \$	SAVE GEN				
		= 342	MOV @R1, A				
		= 343 \$	RESTORE				
		= 344	EXITM				

LOC	OBJ	LINE	SOURCE STATEMENT	TRANSLATE	SYMBOL	LOC	OBJ
..		= 345	ENDIF				
..		= 346	IF ?&DEST EQ 0 OR ?&DEST EQ 1				
..		= 347	\$ SAVE GEN				
..		= 348	MOV DEST, A				
..		= 349	\$ RESTORE				
..		= 350	EXITM				
..		= 351	ENDIF				
..		= 352	ERROR 1				
..		= 353	ENDM				
..		= 354	;				
..		= 355	;;?FORM4 MACRO FOR GENERALIZING 'MOV A, SRC' INSTRUCTION				
..		= 356	?FORM4 MACRO SRC				
..		= 357	IF ?&SRC EQ 2				
..		= 358	\$ SAVE GEN				
..		= 359	MOV R1, #SRC				
..		= 360	MOV R, R1				
..		= 361	\$ RESTORE				
..		= 362	EXITM				
..		= 363	ENDIF				
..		= 364	IF ?&SRC EQ 0 OR ?&SRC EQ 1				
..		= 365	\$ SAVE GEN				
..		= 366	MOV R, SRC				
..		= 367	\$ RESTORE				
..		= 368	EXITM				
..		= 369	ENDIF				
..		= 370	IF ?&SRC EQ 3				
..		= 371	\$ SAVE GEN				
..		= 372	MOV R, #SRC				
..		= 373	\$ RESTORE				
..		= 374	EXITM				
..		= 375	ENDIF				
..		= 376	ERROR 1				
..		= 377	ENDM				
..		= 378	;				
..		= 379	;;?FORM5 MACRO FOR GENERALIZING MOVING A CONSTANT INTO A VARIABLE				
..		= 380	?FORM5 MACRO DEST, CONST				
..		= 381	IF ?&DEST EQ 0 OR ?&DEST EQ 1 OR ?&DEST EQ 4				
..		= 382	\$ SAVE GEN				
..		= 383	MOV DEST, #CONST				
..		= 384	\$ RESTORE				
..		= 385	EXITM				
..		= 386	ENDIF				
..		= 387	IF ?&DEST EQ 2				
..		= 388	\$ SAVE GEN				
..		= 389	MOV R1, #DEST				
..		= 390	MOV R1, #CONST				
..		= 391	\$ RESTORE				
..		= 392	EXITM				
..		= 393	ENDIF				
..		= 394	ERROR 1				
..		= 395	ENDM				
..		= 396	;				
..		= 397	;;MMOV MACRO GENERALIZED MOVE FROM SRC TO DEST				
..		= 398	MMOV MACRO DEST, SRC				
..		= 399	IF ?&SRC EQ 3				

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUCTION	OPERANDS	LOC	OBJ
-		= 400	?FORM5 DEST, SRC	MOV	DEST, SRC	-	
-		= 401	EXITM	EXITM		-	
-		= 402	ENDIF			-	
-		= 403	IF ?DEST EQ 4	IF	DEST EQ 4	-	
-		= 404	?FORM1 MOV, SRC	MOV	SRC	-	
-		= 405	EXITM	EXITM		-	
-		= 406	ENDIF			-	
-		= 407	IF ?SRC EQ 4	IF	SRC EQ 4	-	
-		= 408	?FORM2 DEST	MOV	DEST	-	
-		= 409	EXITM	EXITM		-	
-		= 410	ENDIF			-	
-		= 411	?FORM1 MOV, SRC	MOV	SRC	-	
-		= 412	?FORM2 DEST	MOV	DEST	-	
-		= 413	ENDM			-	
-		= 414	?BINOP MACRO GENERALIZES ARITHMETIC AND LOGICAL OPERATIONS			-	
-		= 415	?BINOP MACRO OPCODE, DEST, SRC			-	
-		= 416	IF ?DEST EQ 4	IF	DEST EQ 4	-	
-		= 417	?FORM1 OPCODE, SRC	MOV	SRC	-	
-		= 418	EXITM	EXITM		-	
-		= 419	ENDIF			-	
-		= 420	IF ?SRC EQ 4	IF	SRC EQ 4	-	
-		= 421	?FORM1 OPCODE, DEST	MOV	DEST	-	
-		= 422	?FORM2 DEST	MOV	DEST	-	
-		= 423	EXITM	EXITM		-	
-		= 424	ENDIF			-	
-		= 425	?FORM1 MOV, SRC	MOV	SRC	-	
-		= 426	?FORM1 OPCODE, DEST	MOV	DEST	-	
-		= 427	?FORM2 DEST	MOV	DEST	-	
-		= 428	ENDM			-	
-		= 429	:MADD MACRO FOR GENERALIZING ADD INSTRUCTION			-	
-		= 430	:MADD MACRO DEST, SRC			-	
-		= 431	?BINOP ADD, DEST, SRC			-	
-		= 432	ENDM			-	
-		= 433				-	
-		= 434	:MADD MACRO FOR GENERALIZING ADDC INSTRUCTION			-	
-		= 435	:MADD MACRO DEST, SRC			-	
-		= 436	?BINOP ADDC, DEST, SRC			-	
-		= 437	ENDM			-	
-		= 438				-	
-		= 439	:MANL MACRO FOR GENERALIZING ANL INSTRUCTION			-	
-		= 440	:MANL MACRO DEST, SRC			-	
-		= 441	?BINOP ANL, DEST, SRC			-	
-		= 442	ENDM			-	
-		= 443				-	
-		= 444	:MORL MACRO FOR GENERALIZING ORL INSTRUCTION			-	
-		= 445	:MORL MACRO DEST, SRC			-	
-		= 446	?BINOP ORL, DEST, SRC			-	
-		= 447	ENDM			-	
-		= 448				-	
-		= 449	:MXRL MACRO FOR GENERALIZING XRL INSTRUCTION			-	
-		= 450	:MXRL MACRO DEST, SRC			-	
-		= 451	?BINOP XRL, DEST, SRC			-	
-		= 452	ENDM			-	
-		= 453				-	
-		= 454	:MXCH MACRO FOR GENERALIZING XCH INSTRUCTION			-	

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUMENT	LOC	OBJ
		= 455	MMCH MACRO DEST, SRC			
		= 456	?BINOP XCH, DEST, SRC			
		= 457	ENDM			
		= 458 ;				
		= 459 ?UNARY MACRO OPCODE, DEST				
		= 460	?FORM1 MOV, DEST			
		= 461 \$SAVE GEN				
		= 462	OPCODE A			
		= 463 \$RESTORE				
		= 464	?FORM3 DEST			
		= 465	ENDM			
		= 466 ;				
		= 467 MINC MACRO DEST				
		= 468	?UNARY INC, DEST			
		= 469	ENDM			
		= 470 ;				
		= 471 MDEC MACRO DEST				
		= 472	?UNARY DEC, DEST			
		= 473	ENDM			
		= 474 ;				
		= 475 MDJNZ MACRO DEST, ADDR				
		= 476	?UNARY DEC, DEST			
		= 477 \$SAVE GEN				
		= 478	JNZ ADDR			
		= 479 \$RESTORE				
		= 480	ENDM			
		= 481 ;				
		= 482 MRL MACRO DEST				
		= 483	?UNARY RL, DEST			
		= 484	ENDM			
		= 485 ;				
		= 486 MRR MACRO DEST				
		= 487	?UNARY RR, DEST			
		= 488	ENDM			
		= 489 ;				
		= 490 MRRC MACRO DEST				
		= 491	?UNARY RRC, DEST			
		= 492	ENDM			
		= 493 ;				
		= 494 MRLC MACRO DEST				
		= 495	?UNARY RLC, DEST			
		= 496	ENDM			
		= 497 ;				
		= 498 \$EJECT				

1

LOC	OBJ	LINE	SOURCE STATEMENT	GENERATE SOURCE	LINE	LOC
		499 ;				
		500 ;	=====			
		501 ;	=====			
		502 ;	BEGINNING OF PROGRAM PROPER			
		503 ;	=====			
		504 ;	=====			
		505 ;				
		506 ;				
		507 ;	*****			
		508 ;				
		509 ;	ALLOCATION OF MP I/O PORTS:			
		510 ;				
		511 ;	*****			
		512 ;				
		513 ;	BUS ;USED FOR BIDIRECTIONAL ADDRESS AND DATA TRANSFERS			
		514 ;	P1 ;USED AS INDIVIDUAL CONTROL OUTPUTS AND BREAK LOGIC			
		515 ;	P2 ;HIGH-ORDER ADDRESS AND ADDRESS SPACE SELECTION			
		516 ;				
000E		517 PDIGIT EQU P7	;USED TO ENABLE CHARACTERS AND STROBE ROWS OF KEYBOARD			
000D		518 PSEGLI EQU P6	;USED TO TURN ON HI SEGMENTS OF CURRENTLY ENABLED DIGIT			
000C		519 PSEGLO EQU P5	;PORT FOR LOWER FOUR SEGMENTS			
000B		520 PINPUT EQU P4	;PORT USED TO SCAN FOR KEY CLOSURES			
		521 ;				
		522 ;	*****			
		523 ;				
		524 ;	INDIVIDUAL PINS OF PORT 1 USED AS FOLLOWS:			
		525 ;				
		526 ;	*****			
		527 ;				
0001		528 ENDRAM EQU 0000001D	;P10 - HI ENABLES BREAK ON BREAK RAM OUTPUT SIGNAL			
0002		529 ENBLNK EQU 00000010B	;P11 - HI ENABLES BREAK ON RD OR WR TO LINK BY EP			
		530 ;	(NOTE: P11 & P10 BOTH HI ENABLES			
		531 ;	BREAK ON ANY EP INSTRUCTION CYCLE)			
0004		532 EPSSTP EQU 00000100B	;P12 - LO FORCES EP SS INPUT LOW			
		533 ;	HI GATES BREAKPOINT FLIP-FLOP TO EP SS INPUT			
0008		534 CLRBPFF EQU 00001000B	;P13 - LO CLEARS BREAK FLIP-FLOP			
		535 ;	AND ENABLES WR CONTROL TO BREAKPOINT RAM			
0010		536 EPRSET EQU 00010000B	;P14 - HI RESETS EP			
0020		537 MODOUT EQU 00100000B	;P15 - LO WHEN EP IS EXECUTING USER PROGRAM			
		538 ;	HI WHEN EP FROZEN OR RUNNING OVERLAYS			
0040		539 TTYOUT EQU 01000000B	;P16 - SERIAL OUTPUT TO TTY OR CRT			
		540 ;	P17 - UNUSED			
		541 ;				
		542 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	PC	DISP
		543	*****		012	
		544	;		012	
		545	INDIVIDUAL PINS OF PORT 2 USED AS FOLLOWS		012	
		546	;		012	
		547	*****		012	
		548	;		012	
		549	P23-P20 ;ADR11-ADR8 FOR ACCESSING PROGRAM OR DATA RAM ARRAY		012	
0010		551 M0 EQU 00010000B ;P24 -- MEMORY MATRIX CONTROL PIN 0		012	0000	
0020		552 M1 EQU 00100000B ;P25 -- MEMORY MATRIX CONTROL PIN 1		012		
0040		553 MPUSEL EQU 01000000B ;P26 -- HIGH WHEN MP IN CONTROL OF COMMON MEM ARRAY,		012		
		554 ; LOW WHEN EP IN CONTROL		012		
0060		555 EXPMON EQU 10000000B ;P27 -- JUMPERED TO GROUND FOR STANDARD MONITOR,		012		
		556 ; FLOATING WHEN EXPANSION MONITOR PRESENT.		012		
		557 ;		012		
		558 ;		012		
		559 ; WHEN MP IN CONTROL OF MEMORY MATRIX M1-M0 USED AS FOLLOWS		012		
		560 ;		012		
		561 ; M1 M0 MODE		012		
		562 ; 0 0 PROGRAM RAM ARRAY ENABLED FOR READ & WRITE		012		
		563 ; 0 1 DATA RAM ARRAY ENABLED FOR READ & WRITE		012		
		564 ; 1 X LINK REGISTER ENABLED FOR READ, RAM ARRAYS DISABLED.		012		
		565 ; (NOTE: LINK REGISTER ALWAYS ENABLED FOR MP WRITES)		012		
		566 ;		012		
		567 ; WHEN EP IN CONTROL OF MATRIX M1-M0 USED AS FOLLOWS		012		
		568 ;		012		
		569 ; M1 M0 MODE		012		
		570 ; 0 X EP PSEN FETCHES FROM LINK REGISTER (USED TO FORCE OPCODES)		012		
		571 ; 1 0 EP PSEN FETCHES FROM PROGRAM RAM ARRAY,		012		
		572 ; EP RD & WR CONTROL DATA RAM ARRAY.		012		
		573 ; 1 1 EP PSEN FETCHES FROM PROGRAM RAM ARRAY,		012		
		574 ; RD & WR CONTROL LINK REGISTER.		012		
		575 ;		012		
		576 \$EJECT		012		

LOC	OBJ	LINE	SOURCE STATEMENT	LOC	OBJ
		577	*****		
		578	*****		
		579	*****		
		580	SYSTEM CONSTANT DEFINITIONS		
		581	*****		
		582	*****		
		583	*****		
0008		584	DECLARE CHARNO.CONST ,NUMBER OF DIGITS IN DISPLAY AND ROWS OF KEYS		
		588	CHARNO EQU 8		
		599	*****		
0004		600	DECLARE NCOLS.CONST ,LESSER DIMENSION OF KEYBOARD MATRIX		
		614	NCOLS EQU 4		
		615	*****		
0008		616	DECLARE DEBNCN.CONST ,NUMBER OF SUCCESSIVE SCANS BEFORE KEY CLOSURE ACCEPTED		
		630	DEBNCN EQU 8		
		631	*****		
0017		632	DECLARE OVSZ.CONST ,SIZE OF LARGEST MINI-MONITOR OVERLAY FOR EP		
		640	OVSZ EQU 23		
		647	*****		
0010		648	DECLARE BUFLN.CONST ,LENGTH OF HEX FORMAT XMIT BUFFER (MAX RECORD LENGTH)		
		662	BUFLN EQU 16		
		663	*****		
		664	*****		
		665	*****		
		666	UTILITY CONSTANT DECLARATIONS		
		667	*****		
		668	*****		
		669	*****		
0000		670	DECLARE ZERO.CONST		
		684	ZERO EQU 0		
		685	DECLARE PLUS1.CONST		
0001		699	PLUS1 EQU 1		
		700	DECLARE PLUS3.CONST		
0003		714	PLUS3 EQU 3		
		715	DECLARE NEG1.CONST		
FFFF		729	NEG1 EQU -1		
		730	*****		
		731	\$EJECT		

1

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	LOC	OBJ
		957				
		958	*****			
		959				
		960	DATA RAM ALLOCATION			
		961				
		962	*****			
		963				
0020		964	DECLARE EPACC,RAM ; STORAGE IN MP FOR EP ACCUMULATOR			
		969+	EPACC EQU 32			
0021		973	DECLARE EPPSW,RAM ; STORAGE IN MP FOR EP PROGRAM STATUS WORD			
		970+	EPPSW EQU 33			
0022		982	DECLARE EPTMR,RAM ; STORAGE IN MP FOR EP TIMER/COUNTER REGISTER			
		987+	EPTMR EQU 34			
		991	DECLARE EPR0,RAM ; STORAGE IN MP FOR EP REGISTER 0 OF BANK 0			
0023		996+	EPR0 EQU 35			
		1000	DECLARE EPPCLO,RAM ; STORAGE IN MP FOR LOW BYTE OF EP PROGRAM COUNTER			
0024		1005+	EPPCLO EQU 36			
		1009	DECLARE EPPCHI,RAM ; STORAGE IN MP FOR HIGH NIBBLE OF EP PROGRAM COUNTER			
0025		1014+	EPPCHI EQU 37			
		1018	DECLARE HBITLO,RAM ; PARAMETER 1 FOR SERIAL LINK DATA RATE GENERATOR			
0026		1023+	HBITLO EQU 38			
		1027	DECLARE HBITHI,RAM ; PARAMETER 2 FOR SERIAL LINK DATA RATE GENERATOR			
0027		1032+	HBITHI EQU 39			
		1036	DECLARE DSPTIM,RAM ; PARAMETER FOR AUTO-STEP AND AUTO-BREAK SEQUENCING RATE			
0028		1041+	DSPTIM EQU 40			
		1045	DECLARE VERSNO,RAM ; MONITOR VERSION NUMBER			
0029		1050+	VERSNO EQU 41			
		1054	DECLARE HREGA,RAM ; (UNUSED)			
002A		1059+	HREGA EQU 42			
		1063	DECLARE HREGD,RAM ; (UNUSED)			
002B		1068+	HREGD EQU 43			
		1072	DECLARE HRECC,RAM ; (UNUSED)			
002C		1077+	HRECC EQU 44			
		1081	DECLARE HREGD,RAM ; (UNUSED)			
002D		1086+	HREGD EQU 45			
		1090	DECLARE HREGF,RAM ; (UNUSED)			
002E		1095+	HREGF EQU 46			
		1099	DECLARE HREGF,RAM ; (UNUSED)			
002F		1104+	HREGF EQU 47			
		1108	DECLARE SMALO,RAM ; PRIMARY COMMAND STARTING MEMORY ADDRESS (LOW BYTE)			
0030		1113+	SMALO EQU 48			
		1117	DECLARE SMAHI,RAM ; PRIMARY COMMAND STARTING MEMORY ADDRESS (HIGH BYTE)			
0031		1122+	SMAHI EQU 49			
		1126	DECLARE EMALO,RAM ; PRIMARY COMMAND ENDING MEMORY ADDRESS (LOW BYTE)			
0032		1131+	EMALO EQU 50			
		1135	DECLARE EMAHI,RAM ; PRIMARY COMMAND ENDING MEMORY ADDRESS (HIGH BYTE)			
0033		1140+	EMAH1 EQU 51			
		1144	DECLARE MEMLO,RAM ; THIRD PARSELR PARAMETER & HEX RECORD ADDRESS (LOW)			
0034		1149+	MEMLO EQU 52			
		1153	DECLARE MEMHI,RAM ; THIRD PARSELR PARAMETER & HLX RECORD ADDRESS (HIGH)			
0035		1158+	MEMHI EQU 53			
		1162	DECLARE BCODE,RAM ; PRIMARY COMMAND NUMBER FROM PARSER TABLES (0-9)			
0036		1167+	BCODE EQU 54			
		1171	DECLARE TYPE,RAM ; PRIMARY COMMAND MODIFIER/OPTION (0-5)			
0037		1176+	TYPE EQU 55			

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTEENTH 30902	THIRTY	LOC	OBJ
		1180	DECLARE NUMCON, RAM ; MAX NUMBER OF PARAMETERS ALLOWED FOR SELECTED COMMAND				
0038		1185+	NUMCON EQU 56				
		1189	DECLARE OPTION, RAM ; INDEX POINTER USED IN SEARCHING PARSER TABLES				
0039		1194+	OPTION EQU 57				
		1198	DECLARE NEXTPL, RAM ; CHARACTER POSITION FOR DISPLAY UTILITIES TO WRITE NEXT				
003A		1203+	NEXTPL EQU 58				
		1207	DECLARE KDBUF, RAM ; POSITION OF KEY DEBOUNCE BY SCANNING SUBROUTINE				
003B		1212+	KDBUF EQU 59				
		1216	DECLARE KEYLOC, RAM ; INCREMENTED AS SUCCESSIVE KEY LOCATIONS SCANNED				
003C		1221+	KEYLOC EQU 60				
		1225	DECLARE NREPTS, RAM ; KEEPS TRACK OF SUCCESSIVE READS OF SAME KEYSTROKE				
003D		1230+	NREPTS EQU 61				
		1234	DECLARE ASAVL, RAM ; HOLDS ACCUMULATOR VALUE DURING SERVICE ROUTINE				
003E		1239+	ASAVE EQU 62				
		1243	DECLARE RDELAY, RAM ; COUNTER DECREMENTED WHEN AUTO-STEP DELAY IN PROGRESS				
003F		1248+	RDELAY EQU 63				
		1252	DECLARE STRTMP, RAM ; INDEX POINTER FOR DISPLAY CHARACTER STRING ACCESSING				
0040		1257+	STRTMP EQU 64				
		1261	DECLARE BUFCNT, RAM ; COUNT OF DATA BYTES IN HEX FORMAT RECORD BUFFER				
0041		1266+	BUFCNT EQU 65				
		1270	DECLARE RECTYP, RAM ; TYPE OF HEX FORMAT RECORD (0 OR 1)				
0042		1275+	RECTYP EQU 66				
		1279	DECLARE B, RAM ; BIT COUNTER FOR ASCII SERIAL I/O UTILITY SUBROUTINES				
0043		1284+	B EQU 67				
		1288	DECLARE REGC, RAM ; CHARACTER BEING SHIFTED DURING SERIAL I/O PROCESS				
0044		1293+	REGC EQU 68				
		1297	DECLARE H, RAM ; COUNTER IN SOFTWARE DELAY DATA RATE GENERATOR				
0045		1302+	H EQU 69				
		1306	;				
		1307	MBLOCK SEGMAP, CHARNO ; REGISTER ARRAY FOR DISPLAY PATTERNS				
0046		1311+	SEGMAP EQU 70				
		1314	;				
		1315	MBLOCK OVBUF, OVSZ ; LOW-ORDER USER PROGRAM DURING MINI-MONITOR OVERLAYS				
004E		1319+	OVBUF EQU 78				
		1322	;				
		1323	MBLOCK HEXBUF, BUFLN ; ALLOCATE BLOCK OF RAM FOR USE AS HEX RECORD BUFFER				
0065		1327+	HEXBUF EQU 101				
		1330	;				
		1331	\$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	LOC	OBJ
		1332	DATADLK 40		
0300		1337+	ORG 768		
		1341	INVALS TABLE OF CONSTANTS TO BE LOADED INTO MP INTERNAL RAM VARIABLES		
		1342	AS PART OF SYSTEM INITIALIZATION PROCEDURE:		
		1343			
		1344	INITIAL VALUE VARIABLE TYPE		
		1345	=====		
0300 00		1346	INVALS: DB 00H ; ROTPAT KB1		
0301 00		1347	DB 00H ; ROTCNT RB1		
0302 00		1348	DB 00H ; LASTKY RB1		
0303 00		1349	DB CHARNO ; CURDIG RB1		
0304 00		1350	DB 00H ; KEYFLG RB1		
0305 00		1351	DB 00H ; <REG7> RB1		
0306 00		1352	DB 00H ; EPACC KHM		
0307 01		1353	DB 01H ; EPPSW RAM		
0308 00		1354	DB 00H ; EPTMR RAM		
0309 00		1355	DB 00H ; EPR0 RAM		
030A 00		1356	DB 00H ; EPPCLO RAM		
030B 00		1357	DB 00H ; EPPCHI RAM		
030C 93		1358	DB 93H ; HBITLE RAM		
030D 04		1359	DB 04H ; HBITHI RAM		
030E 20		1360	DB 20H ; DSPTIM RAM		
030F 25		1361	DB 25H ; VERSNO RAM		
0310 00		1362	DB 00H ; HREGA RAM		
0311 00		1363	DB 00H ; HREGB RAM		
0312 00		1364	DB 00H ; HREGC RAM		
0313 00		1365	DB 00H ; HREGD RAM		
0314 00		1366	DB 00H ; HREG E RAM		
0315 00		1367	DB 00H ; HREGF RAM		
0316 00		1368	DB 00H ; SMALO RAM		
0317 00		1369	DB 00H ; SMAHI RAM		
0318 FF		1370	DB 0FFH ; EMAILO RAM		
0319 0F		1371	DB 0FHH ; EMAHI RAM		
031A 00		1372	DB 00H ; MEMLO RAM		
031B 00		1373	DB 00H ; MEMHI RAM		
031C 00		1374	DB 00H ; BCODE RAM		
031D 04		1375	DB 04H ; TYPE RAM		
031E 01		1376	DB 01H ; NUMCON RAM		
031F 00		1377	DB 00H ; OPTION RAM		
0320 00		1378	DB CHARNO ; NEXTPL RAM		
0321 FF		1379	DB 0FFH ; KBDDBUF RAM		
0322 00		1380	DB 00H ; KEYLOC RAM		
0023		1381	NOVALS EQU \$-INVALS		
		1382	SIZECHK		
0023		1385+	SIZE SET 35		
		1386+			
		1387+	*****		
		1396	\$EJECT		

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	LOC	OBJ
		1397 \$	INCLUDE(:F0:PARSER.MOD)			
		=1398	CODEBLK 45			
0000		=1403+	ORG 0			
		=1407	INIT: INITIALIZES PROCESSOR REGISTERS			
		=1408	AND RAM LOCATIONS TO DEFINED VALUES.			
0000 C5		=1409	INIT: SEL R00			
0001 BF00		=1410	MOV XPCODE, #0			
0003 74D1		=1411	CALL XPTST			
0005 27		=1412	CLR A			
0006 3D		=1413	MOVD PSEGLO, A			
0007 3E		=1414	MOVD PSEGLI, A			
0008 D81A		=1415	MOV R0, #1AH ; START AT RD1 (REG2) = RAM LOC 1AH			
000A B923		=1416	MOV R1, #LOW NOVALS			
000C BA00		=1417	MOV R2, #LOW INVALS			
000E FA		=1418	INITLP: MOV A, R2			
000F E3		=1419	MOVP3 A, 0A			
0010 A0		=1420	MOV 0A0, A			
0011 18		=1421	INC R0			
0012 1A		=1422	INC K2			
0013 E90E		=1423	DJNZ R1, INITLP			
0015 55		=1424	STRT T			
0016 744F		=1425	CALL EPRK			
0018 180B		=1426	MOV R0, #LOW(OV1BAS+OVSIZE)			
001A 746A		=1427	CALL OVLOAD			
001C 54E5		=1428	CALL COMFIL			
001E B937		=1429	MOV R1, #TYPE			
0020 11		=1430	INC 0A1			
0021 34F2		=1431	CALL INCSMA			
0023 54E5		=1432	CALL COMFIL			
0025 99E1		=1433	ANL PL, #(NOT EPRSET) ; REMOVE EP RESET SIGNAL			
0027 0429		=1434	JMP MAIN			
		=1435				
		=1436	SIZECHK			
0029		=1439+	SIZE SET 41			
		=1440+				
		=1441+	*****			
		=1450	\$EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	THROWING SOURCE	LINE	LOC	OBJ
=1451 ;				INCLUDE 'P: PAPER.M00'	=1451 ;		
=1452 ;			KEYBOARD LAYOUT:	COORDINATE	=1452 ;		
=1453 ;			=====	0	=1453 ;		
=1454 ;				INITIALISES PROGRAM REGISTERS	=1454 ;		
=1455 ;					=1455 ;		
=1456 ;	!	!!	!!	!!	=1456 ;	23	0000
=1457 ;	!	LIST	!!GO/RESET!! GO	!!EXAM/CHN!!	=1457 ;	0000	0000
=1458 ;	!	!!	!!	!!	=1458 ;	0000	0000
=1459 ;					=1459 ;	0000	0000
=1460 ;					=1460 ;	0000	0000
=1461 ;	!		!!PROG BRK!!	!!PROG MEM!!	=1461 ;	0000	0000
=1462 ;	!	UPLOAD	!!	!!	=1462 ;	0000	0000
=1463 ;	!		!!AUTO STP!!	!!SING STP!!	=1463 ;	0000	0000
=1464 ;					=1464 ;	0000	0000
=1465 ;					=1465 ;	0000	0000
=1466 ;	!		!!DATA BRK!!	!!DATA MEM!!	=1466 ;	0000	0000
=1467 ;	!	DNLOAD	!!	!!	=1467 ;	0000	0000
=1468 ;	!		!!AUTO BRK!!	!!WITH BRK!!	=1468 ;	0000	0000
=1469 ;					=1469 ;	0000	0000
=1470 ;					=1470 ;	0000	0000
=1471 ;	!	!!	!!	!!	=1471 ;	0000	0000
=1472 ;	!	FILL	!!HARD REG!!	!!NEXT/,	=1472 ;	0000	0000
=1473 ;	!	!!	!!	!!	=1473 ;	0000	0000
=1474 ;					=1474 ;	0000	0000
=1475 ;					=1475 ;	0000	0000
=1476			\$JECT		=1476	0000	0000

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	DISPATCH	LOC	OBJ
		=1477 ;					
		=1478 ;	THE FOLLOWING EQUATES DETERMINES HOW THE PARSER INTERPRETS				
		=1479 ;	VALUES RETURNED BY THE KEYBOARD SCANNING INPUT ROUTINE				
		=1480 ;	WHEN THE VARIOUS KEYS OF THE KEYBOARD ARE PRESSED				
		=1481 ;					
		=1482 ;					
		=1483 ;	KEY0 EQU 00H VALUE RETURNED FOR EACH KEY OF KEYBOARD MATRIX				
		=1484 ;	KEY1 EQU 01H BY KEYBOARD SCANNING SUBROUTINE "KBDIN"				
		=1485 ;	KEY2 EQU 02H				
		=1486 ;	KEY3 EQU 03H				
		=1487 ;	KEY4 EQU 04H				
		=1488 ;	KEY5 EQU 05H				
		=1489 ;	KEY6 EQU 06H				
		=1490 ;	KEY7 EQU 07H				
		=1491 ;	KEY8 EQU 08H				
		=1492 ;	KEY9 EQU 09H				
		=1493 ;	KEYA EQU 0AH				
		=1494 ;	KEYD EQU 0BH				
		=1495 ;	KEYC EQU 0CH				
		=1496 ;	KEYD EQU 0DH				
		=1497 ;	KEYE EQU 0EH				
		=1498 ;	KEYF EQU 0FH				
0010		=1499	KEYFIL EQU 10H ;[FILL COMMAND]				
0012		=1500	KEYNXT EQU 12H ;[NEXT/]				
0013		=1501	KEYEND EQU 13H ;[END/]				
0014		=1502	KEYREL EQU 14H ;[DOWNLOAD COMMAND]				
0015		=1503	KEYPAT EQU 15H ;[AUTOBREAK MODIFIER]				
0016		=1504	KEYDM EQU 16H ;[DATA MEMORY MODIFIER]				
0017		=1505	KEYCLR EQU 17H ;[CLEAR/PREVIOUS]				
0018		=1506	KEYREC EQU 18H ;[UPLOAD COMMAND]				
0019		=1507	KEYTRA EQU 19H ;[AUTOSTEP MODIFIER]				
001A		=1508	KEYPM EQU 1AH ;[PROGRAM MEMORY MODIFIER]				
001B		=1509	KEYREG EQU 1BH ;[REGISTER MEMORY MODIFIER]				
001C		=1510	KEYLST EQU 1CH ;[FORMATTED DATA OUTPUT COMMAND]				
001D		=1511	KGORES EQU 1DH ;[GO FROM RESET STATE COMMAND]				
001E		=1512	KEYGO EQU 1EH ;[GO COMMAND]				
001F		=1513	KEYMOD EQU 1FH ;[EXAMINE/MODIFY COMMAND]				
000B		=1514	KSETB EQU 0BH ;[SET BREAKPOINT COMMAND]				
000C		=1515	KCLRB EQU 0CH ;[CLEAR BREAKPOINT COMMAND]				
		=1516 ;					
		=1517 ;					
0019		=1518	PBRK EQU 19H ;[PROGRAM BREAKPOINT MEMORY MODIFIER]				
0015		=1519	DBRK EQU 15H ;[DATA BREAKPOINT MEMORY MODIFIER]				
0011		=1520	RINT EQU 11H ;[HARDWARE REGISTER MEMORY MODIFIER]				
001B		=1521	NOBRK EQU 1BH ;[WITHOUT BREAKPOINTS MODIFIER]				
001C		=1522	WBRK EQU 1CH ;[WITH BREAKPOINTS ENABLED MODIFIER]				
001A		=1523	SING EQU 1AH ;[SINGLE STEP MODIFIER]				
		=1524 ;					
		=1525	\$EJECT				

1

LOC	OBJ	LINE	SOURCE STATEMENT	LINE	LOC
0059	1C	=1625	INC ITMP		
005A	FC	=1626	MOV R, ITMP		
005B	E3	=1627	MOV R, #0 ; GET OPTION POINTER		
		=1628	MMOV OPTION, R		
005C	B939	=1641+	MOV R1, #OPTION		
005E	A1	=1642+	MOV @R1, R		
005F	1C	=1646	INC ITMP		
0060	FC	=1647	MOV R, ITMP		
0061	E3	=1648	MOV R, #0 ; GET NO OF PARAMETERS		
		=1649	MMOV NUMCON, R		
0062	B938	=1662+	MOV R1, #NUMCON		
0064	R1	=1663+	MOV @R1, R		
		=1667 ;			
		=1668 ;	PARAMETER_BUFFER(0)=0		
		=1669 ;			
0065	B90C	=1670	MOV R1, #6 ; EACH PARAM IS 2 BYTES		
0067	B830	=1671	MOV R0, #SMALO ; START OF PARAM BUFFERS		
0069	B000	=1672 MAINB:	MOV @R0, #00H		
006B	18	=1673	INC R0		
006C	E969	=1674	DJNZ R1, MAINB		
006E	14EC	=1675	CALL INKEY		
		=1676 ;			
		=1677 ;	WHILE KEY>MEM(OPTION+TYPE)[6-0] DO		
		=1678 ;	IF MEM(OPTION+TYPE)[7]=1 GOTO MAIND1		
		=1679 ;	TYPE = TYPE+1		
		=1680 ;	ENDWHILE		
		=1681 ;			
		=1682	MMOV ITMP, OPTION		
0070	B939	=1690+	MOV R1, #OPTION		
0072	F1	=1699+	MOV R, @R1		
0073	0C	=1712+	MOV ITMP, R		
0074	1C	=1715	INC ITMP		
		=1716 MAINC1:	MMOV R, ITMP		
0075	FC	=1732+	MOV R, ITMP		
0076	E3	=1736	MOV R, #0		
0077	97	=1737	CLR C		
0078	F7	=1738	RLC R		
0079	77	=1739	RR R ; STRIP BIT SEVEN INTO CARRY		
007A	DE	=1740	XRL R, KEY		
007B	C693	=1741	JZ MAIND		
007D	F687	=1742	JC MAIND1		
		=1743	INCR TYPE		
007F	B937	=1748+	MOV R1, #TYPE		
0081	F1	=1749+	MOV R, @R1		
0082	17	=1753+	INC R		
0083	A1	=1758+	MOV @R1, R		
0084	1C	=1761	INC ITMP		
0085	0475	=1762	JMP MAINC1		
		=1763 ;			
		=1764 ;	MODIFIER NOT FOUND SO RESET TYPE INDEX TO DEFAULT CASE (ZERO).		
		=1765 ;			
		=1766 MAIND1:	MMOV TYPE, ZERO		
0087	B937	=1777+	MOV R1, #TYPE		
0089	B100	=1778+	MOV @R1, #ZERO		
		=1782	MMOV R, OPTION		

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
0088	B939	=1791+	MOV R1, #OPTION	MOV R1, #OPTION	0088	B939
008D	F1	=1792+	MOV A, @R1	MOV A, @R1	008D	F1
008E	E3	=1796	MOV P3, A, @R1	MOV P3, A, @R1	008E	E3
008F	3404	=1797	CALL OUTMSG	CALL OUTMSG	008F	3404
0091	049E	=1798	JMP MAINB0	JMP MAINB0	0091	049E
		=1799 ;				
		=1800 ;	CALL OUTPUT_MESSAGE(MODIFIER)	CALL OUTPUT_MESSAGE(MODIFIER)		
		=1801 MAIND:	MMOV A, OPTION	MMOV A, OPTION		
0093	B939	=1810+	MOV R1, #OPTION	MOV R1, #OPTION	0093	B939
0095	F1	=1811+	MOV A, @R1	MOV A, @R1	0095	F1
0096	E3	=1815	MOV P3, A, @R1	MOV P3, A, @R1	0096	E3
		=1816	MADD A, TYPE	MADD A, TYPE		
0097	B937	=1822+	MOV R1, #TYPE	MOV R1, #TYPE	0097	B937
0099	G1	=1823+	ADD A, @R1	ADD A, @R1	0099	G1
009A	3404	=1827	CALL OUTMSG	CALL OUTMSG	009A	3404
009C	14EC	=1828	CALL INPKEY	CALL INPKEY	009C	14EC
		=1829 ;				
009E	BC00	=1830 MAINB0:	MOV ITMP, #0	MOV ITMP, #0	009E	BC00
00A0	2330	=1831 MAINB1:	MOV A, #SMILO	MOV A, #SMILO	00A0	2330
00A2	GC	=1832	ADD A, ITMP	ADD A, ITMP	00A2	GC
00A3	GC	=1833	ADD A, ITMP	ADD A, ITMP	00A3	GC
00A4	A8	=1834	MOV R0, A	MOV R0, A	00A4	A8
00A5	14C0	=1835	CALL INPADK	CALL INPADK	00A5	14C0
00A7	FGBH	=1836	JC CMDINT	JC CMDINT	00A7	FGBH
00A9	1C	=1837	INC ITMP	INC ITMP	00A9	1C
00AA	B938	=1838	MOV R1, #NUMCON	MOV R1, #NUMCON	00AA	B938
00AC	F1	=1839	MOV A, @R1	MOV A, @R1	00AC	F1
00AD	07	=1840	DEC A	DEC A	00AD	07
00AE	A1	=1841	MOV @R1, A	MOV @R1, A	00AE	A1
00AF	C6BA	=1842	JZ CMDINT	JZ CMDINT	00AF	C6BA
00B1	FB	=1843	MOV A, KEY	MOV A, KEY	00B1	FB
00B2	D313	=1844	XRL A, #KEYEND	XRL A, #KEYEND	00B2	D313
00B4	CGBH	=1845	JZ CMDINT	JZ CMDINT	00B4	CGBH
00B6	14EC	=1846	CALL INPKEY	CALL INPKEY	00B6	14EC
00B8	04A0	=1847	JMP MAINB1	JMP MAINB1	00B8	04A0
		=1848 ;				
		=1849 ;	CMDINT ENTER THE COMMAND PROCESSOR WITH:			
		=1850 ;	BASE_CODE=THE MAIN COMMAND TYPE			
		=1851 ;	TYPE=SUBCOMMAND TYPE			
		=1852 ;	PARAMETER(1)=FIRST ADDRESS			
		=1853 ;	PARAMETER(2)=SECOND ADDRESS			
		=1854 ;	PARAMETER(3)=DATA			
00BA	4400	=1855 CMDINT:	JMP IMPLM	JMP IMPLM	00BA	4400
		=1856 ;				
		=1857 ;	MERROR ERROR ENCOUNTERED IN MAIN PARSING ROUTINE			
00BC	BA01	=1858 MERROR:	MOV LDATA, #1	MOV LDATA, #1	00BC	BA01
00BE	249A	=1859	JMP PERROR	JMP PERROR	00BE	249A
		=1860	SIZECHK	SIZECHK		
0097		=1863+ SIZE SET 151			0097	
		=1864+ ;				
		=1865+ ;	*****			
		=1874 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	DISASSEMBLY	LOC	OBJ
		=1945	CODEBLK 130			
0100		=1955+	ORG 256			
		=1959	OUTUTL OUTPUT ONE OF FOUR UTILITY DISPLAY PROMPTS (LEFT JUSTIFIED)			
		=1960	ACCORDING TO ACC CONTENTS (0-3).			
		=1961	OUTCLR CLEAR DISPLAY AND OUTPUT CHARACTER STRING STARTING			
		=1962	AT THE ADDRESS POINTED TO BY BYTE AT ADDRESS IN ACCUMULATOR.			
		=1963	OUTMSG SUBROUTINE TO COPY A STRING OF BIT PATTERNS FROM ROM TO THE			
		=1964	DISPLAY REGISTERS.			
		=1965	STRING SELECTED IS DETERMINED BY ACC WHEN CALLED.			
		=1966	ON ENTERING OUTMSG, ACC CONTENTS ARE USED TO ADDRESS A BYTE IN A			
		=1967	LOOKUP TABLE ON THE CURRENT PAGE WHICH CONTAINS THE ADDRESS OF			
		=1968	A STRING OF SEGMENT PATTERN DATA BYTES TO BE PRINTED ONTO THE			
		=1969	DISPLAY.			
		=1970	THE END OF THE STRING IS INDICATED WHEN BIT7 =1.			
		=1971	CALLS SUBROUTINE 'WDISP'			
		=1972	TO ACTUALLY EFFECT WRITING INTO THE DISPLAY REGISTERS.			
0100	0319	=1973	OUTUTL: ADD A, #STRUTL			
0102	04F1	=1974	OUTCLR: CALL CLEAR			
0104	A3	=1975	OUTMSG: MOVP A, @A			
		=1976	MNOV STRTMP, A			
0105	B940	=1989+	MOV R1, #STRTMP			
0107	A1	=1990+	MOV @R1, A			
		=1994	PRNT2: MNOV A, STRTMP ; LOAD NEXT CHARACTER LOCATION			
0108	B940	=2003+	MOV R1, #STRTMP			
010A	F1	=2004+	MOV A, @R1			
010B	A3	=2008	MOVP A, @A ; LOAD BIT PATTERN INDIRECT			
010C	F217	=2009	JB7 PRNT1			
010E	D4D8	=2010	CALL WDISP ; OUTPUT TO NEXT CHARACTER POSITION			
		=2011	INCR STRTMP ; INDEX POINTER			
0110	D940	=2016+	MOV R1, #STRTMP			
0112	F1	=2017+	MOV A, @R1			
0113	17	=2021+	INC A			
0114	A1	=2026+	MOV @R1, A			
0115	2408	=2029	JMP PRNT2			
0117	C4D8	=2030	PRNT1: JMP WDISP ; DONE			
		=2031				
0019		=2032	STRUTL EQU LOW \$			
0119	31	=2033	DB LOW(DERROR) ; UTILITY MESSAGE 0 ADDRESS			
011A	37	=2034	DB LOW(DSGNON) ; UTILITY MESSAGE 1 ADDRESS			
011B	3E	=2035	DB LOW(DRUN) ; UTILITY MESSAGE 2 ADDRESS			
011C	44	=2036	DB LOW(DBPNT) ; UTILITY MESSAGE 3 ADDRESS			
001D		=2037	STRCOM EQU LOW \$			
011D	46	=2038	DB LOW(DMOD) ; BASIC COMMAND 0 RESPONSE ADDRESS			
011E	49	=2039	DB LOW(DGO) ; BASIC COMMAND 1 RESPONSE ADDRESS			
011F	4B	=2040	DB LOW(DFILL) ; BASIC COMMAND 2 RESPONSE ADDRESS			
0120	4E	=2041	DB LOW(DLST) ; BASIC COMMAND 3 RESPONSE ADDRESS			
0121	51	=2042	DB LOW(DREC) ; BASIC COMMAND 4 RESPONSE ADDRESS			
0122	54	=2043	DB LOW(DREL) ; BASIC COMMAND 5 RESPONSE ADDRESS			
0123	57	=2044	DB LOW(DSB) ; BASIC COMMAND 6 RESPONSE ADDRESS			
0124	5A	=2045	DB LOW(DCB) ; BASIC COMMAND 7 RESPONSE ADDRESS			
0125	5D	=2046	DB LOW(DGR) ; BASIC COMMAND 8 RESPONSE ADDRESS			
0026		=2047	STRMEM EQU LOW \$			
0126	5F	=2048	DB LOW(DPRMEM) ; DATA TYPE MODIFIER 0 RESPONSE ADDRESS			
0127	61	=2049	DB LOW(DDRMEM) ; DATA TYPE MODIFIER 1 RESPONSE ADDRESS			
0128	63	=2050	DB LOW(DRM) ; DATA TYPE MODIFIER 2 RESPONSE ADDRESS			

LOC	OBJ	LINE	SOURCE STATEMENT	LINE	LOC
0129 69		=2051	DB LOW(DINTRG) ; DATA TYPE MODIFIER 3 RESPONSE ADDRESS	4129	
012A 65		=2052	DB LOW(DPRBRK) ; DATA TYPE MODIFIER 4 RESPONSE ADDRESS	412A	
012B 67		=2053	DB LOW(DDABRK) ; DATA TYPE MODIFIER 5 RESPONSE ADDRESS	412B	
002C		=2054	SYRGOC EQU LOW \$	412C	
012C 68		=2055	DB LOW(DNABRK) ; EXECUTION MODE MODIFIER 0	412C	
012D 6D		=2056	DB LOW(DWBRK) ; EXECUTION MODE MODIFIER 1	412D	
012E 6F		=2057	DB LOW(DSS) ; EXECUTION MODE MODIFIER 2	412E	
012F 72		=2058	DB LOW(DPA) ; EXECUTION MODE MODIFIER 3	412F	
0130 75		=2059	DB LOW(DTR) ; EXECUTION MODE MODIFIER 4	4130	
		=2060	;	4130	
		=2061	UTILITY OUTPUT MESSAGES	4131	
		=2062	;	4131	
		=2063	DError:	4132	
0131 79		=2064	DB 01111001D ; "E"	4131	
0132 50		=2065	DB 01010000B ; "R"	4132	
0133 50		=2066	DB 01010000B ; "R"	4133	
0134 5C		=2067	DB 01011100B ; "0"	4134	
0135 50		=2068	DB 01010000B ; "R"	4135	
0136 C0		=2069	DB 11000000B ; "-"	4136	
		=2070	DSGNON:	4137	
0137 00		=2071	DB 00000000B ; " "	4137	
0138 76		=2072	DB 01110110B ; "H"	4138	
0139 6D		=2073	DB 01101101B ; "S"	4139	
013A 79		=2074	DB 01111001D ; "E"	413A	
013B 40		=2075	DB 01000000B ; "-"	413B	
013C 66		=2076	DB 01100110B ; "4"	413C	
013D E7		=2077	DB 11100111B ; "9." (TH)	413D	
		=2078	DRUN:	413E	
013E 00		=2079	DB 00000000B ; " "	413E	
013F 40		=2080	DB 01000000B ; "-"	413F	
0140 50		=2081	DB 01010000B ; "R"	4140	
0141 1C		=2082	DB 00011100B ; "U"	4141	
0142 54		=2083	DB 01010100B ; "N"	4142	
0143 C0		=2084	DB 11000000B ; "-"	4143	
		=2085	DBFNT:	4144	
0144 73		=2086	DB 01110011B ; "P"	4144	
0145 B9		=2087	DB 10111001B ; "C."	4145	
		=2088	\$JECT	4146	
			UTILITY MESSAGE 0 ADDRESS	4147	
			UTILITY MESSAGE 1 ADDRESS	4148	
			UTILITY MESSAGE 2 ADDRESS	4149	
			UTILITY MESSAGE 3 ADDRESS	4150	
			UTILITY MESSAGE 4 ADDRESS	4151	
			UTILITY MESSAGE 5 ADDRESS	4152	
			UTILITY MESSAGE 6 ADDRESS	4153	
			UTILITY MESSAGE 7 ADDRESS	4154	
			UTILITY MESSAGE 8 ADDRESS	4155	
			UTILITY MESSAGE 9 ADDRESS	4156	
			UTILITY MESSAGE 10 ADDRESS	4157	
			UTILITY MESSAGE 11 ADDRESS	4158	
			UTILITY MESSAGE 12 ADDRESS	4159	
			UTILITY MESSAGE 13 ADDRESS	4160	
			UTILITY MESSAGE 14 ADDRESS	4161	
			UTILITY MESSAGE 15 ADDRESS	4162	
			UTILITY MESSAGE 16 ADDRESS	4163	
			UTILITY MESSAGE 17 ADDRESS	4164	
			UTILITY MESSAGE 18 ADDRESS	4165	
			UTILITY MESSAGE 19 ADDRESS	4166	
			UTILITY MESSAGE 20 ADDRESS	4167	
			UTILITY MESSAGE 21 ADDRESS	4168	
			UTILITY MESSAGE 22 ADDRESS	4169	
			UTILITY MESSAGE 23 ADDRESS	4170	
			UTILITY MESSAGE 24 ADDRESS	4171	
			UTILITY MESSAGE 25 ADDRESS	4172	
			UTILITY MESSAGE 26 ADDRESS	4173	
			UTILITY MESSAGE 27 ADDRESS	4174	
			UTILITY MESSAGE 28 ADDRESS	4175	
			UTILITY MESSAGE 29 ADDRESS	4176	
			UTILITY MESSAGE 30 ADDRESS	4177	
			UTILITY MESSAGE 31 ADDRESS	4178	
			UTILITY MESSAGE 32 ADDRESS	4179	
			UTILITY MESSAGE 33 ADDRESS	4180	
			UTILITY MESSAGE 34 ADDRESS	4181	
			UTILITY MESSAGE 35 ADDRESS	4182	
			UTILITY MESSAGE 36 ADDRESS	4183	
			UTILITY MESSAGE 37 ADDRESS	4184	
			UTILITY MESSAGE 38 ADDRESS	4185	
			UTILITY MESSAGE 39 ADDRESS	4186	
			UTILITY MESSAGE 40 ADDRESS	4187	
			UTILITY MESSAGE 41 ADDRESS	4188	
			UTILITY MESSAGE 42 ADDRESS	4189	
			UTILITY MESSAGE 43 ADDRESS	4190	
			UTILITY MESSAGE 44 ADDRESS	4191	
			UTILITY MESSAGE 45 ADDRESS	4192	
			UTILITY MESSAGE 46 ADDRESS	4193	
			UTILITY MESSAGE 47 ADDRESS	4194	
			UTILITY MESSAGE 48 ADDRESS	4195	
			UTILITY MESSAGE 49 ADDRESS	4196	
			UTILITY MESSAGE 50 ADDRESS	4197	
			UTILITY MESSAGE 51 ADDRESS	4198	
			UTILITY MESSAGE 52 ADDRESS	4199	
			UTILITY MESSAGE 53 ADDRESS	4200	
			UTILITY MESSAGE 54 ADDRESS	4201	
			UTILITY MESSAGE 55 ADDRESS	4202	
			UTILITY MESSAGE 56 ADDRESS	4203	
			UTILITY MESSAGE 57 ADDRESS	4204	
			UTILITY MESSAGE 58 ADDRESS	4205	
			UTILITY MESSAGE 59 ADDRESS	4206	
			UTILITY MESSAGE 60 ADDRESS	4207	
			UTILITY MESSAGE 61 ADDRESS	4208	
			UTILITY MESSAGE 62 ADDRESS	4209	
			UTILITY MESSAGE 63 ADDRESS	4210	
			UTILITY MESSAGE 64 ADDRESS	4211	
			UTILITY MESSAGE 65 ADDRESS	4212	
			UTILITY MESSAGE 66 ADDRESS	4213	
			UTILITY MESSAGE 67 ADDRESS	4214	
			UTILITY MESSAGE 68 ADDRESS	4215	
			UTILITY MESSAGE 69 ADDRESS	4216	
			UTILITY MESSAGE 70 ADDRESS	4217	
			UTILITY MESSAGE 71 ADDRESS	4218	
			UTILITY MESSAGE 72 ADDRESS	4219	
			UTILITY MESSAGE 73 ADDRESS	4220	
			UTILITY MESSAGE 74 ADDRESS	4221	
			UTILITY MESSAGE 75 ADDRESS	4222	
			UTILITY MESSAGE 76 ADDRESS	4223	
			UTILITY MESSAGE 77 ADDRESS	4224	
			UTILITY MESSAGE 78 ADDRESS	4225	
			UTILITY MESSAGE 79 ADDRESS	4226	
			UTILITY MESSAGE 80 ADDRESS	4227	
			UTILITY MESSAGE 81 ADDRESS	4228	
			UTILITY MESSAGE 82 ADDRESS	4229	
			UTILITY MESSAGE 83 ADDRESS	4230	
			UTILITY MESSAGE 84 ADDRESS	4231	
			UTILITY MESSAGE 85 ADDRESS	4232	
			UTILITY MESSAGE 86 ADDRESS	4233	
			UTILITY MESSAGE 87 ADDRESS	4234	
			UTILITY MESSAGE 88 ADDRESS	4235	
			UTILITY MESSAGE 89 ADDRESS	4236	
			UTILITY MESSAGE 90 ADDRESS	4237	
			UTILITY MESSAGE 91 ADDRESS	4238	
			UTILITY MESSAGE 92 ADDRESS	4239	
			UTILITY MESSAGE 93 ADDRESS	4240	
			UTILITY MESSAGE 94 ADDRESS	4241	
			UTILITY MESSAGE 95 ADDRESS	4242	
			UTILITY MESSAGE 96 ADDRESS	4243	
			UTILITY MESSAGE 97 ADDRESS	4244	
			UTILITY MESSAGE 98 ADDRESS	4245	
			UTILITY MESSAGE 99 ADDRESS	4246	
			UTILITY MESSAGE 100 ADDRESS	4247	
			UTILITY MESSAGE 101 ADDRESS	4248	
			UTILITY MESSAGE 102 ADDRESS	4249	
			UTILITY MESSAGE 103 ADDRESS	4250	
			UTILITY MESSAGE 104 ADDRESS	4251	
			UTILITY MESSAGE 105 ADDRESS	4252	
			UTILITY MESSAGE 106 ADDRESS	4253	
			UTILITY MESSAGE 107 ADDRESS	4254	
			UTILITY MESSAGE 108 ADDRESS	4255	
			UTILITY MESSAGE 109 ADDRESS	4256	
			UTILITY MESSAGE 110 ADDRESS	4257	
			UTILITY MESSAGE 111 ADDRESS	4258	
			UTILITY MESSAGE 112 ADDRESS	4259	
			UTILITY MESSAGE 113 ADDRESS	4260	
			UTILITY MESSAGE 114 ADDRESS	4261	
			UTILITY MESSAGE 115 ADDRESS	4262	
			UTILITY MESSAGE 116 ADDRESS	4263	
			UTILITY MESSAGE 117 ADDRESS	4264	
			UTILITY MESSAGE 118 ADDRESS	4265	
			UTILITY MESSAGE 119 ADDRESS	4266	
			UTILITY MESSAGE 120 ADDRESS	4267	
			UTILITY MESSAGE 121 ADDRESS	4268	
			UTILITY MESSAGE 122 ADDRESS	4269	
			UTILITY MESSAGE 123 ADDRESS	4270	
			UTILITY MESSAGE 124 ADDRESS	4271	
			UTILITY MESSAGE 125 ADDRESS	4272	
			UTILITY MESSAGE 126 ADDRESS	4273	
			UTILITY MESSAGE 127 ADDRESS	4274	
			UTILITY MESSAGE 128 ADDRESS	4275	
			UTILITY MESSAGE 129 ADDRESS	4276	
			UTILITY MESSAGE 130 ADDRESS	4277	
			UTILITY MESSAGE 131 ADDRESS	4278	
			UTILITY MESSAGE 132 ADDRESS	4279	
			UTILITY MESSAGE 133 ADDRESS	4280	
			UTILITY MESSAGE 134 ADDRESS	4281	
			UTILITY MESSAGE 135 ADDRESS	4282	
			UTILITY MESSAGE 136 ADDRESS	4283	
			UTILITY MESSAGE 137 ADDRESS	4284	
			UTILITY MESSAGE 138 ADDRESS	4285	
			UTILITY MESSAGE 139 ADDRESS	4286	
			UTILITY MESSAGE 140 ADDRESS	4287	
			UTILITY MESSAGE 141 ADDRESS	4288	
			UTILITY MESSAGE 142 ADDRESS	4289	
			UTILITY MESSAGE 143 ADDRESS	4290	
			UTILITY MESSAGE 144 ADDRESS	4291	
			UTILITY MESSAGE 145 ADDRESS	4292	
			UTILITY MESSAGE 146 ADDRESS	4293	
			UTILITY MESSAGE 147 ADDRESS	4294	
			UTILITY MESSAGE 148 ADDRESS	4295	
			UTILITY MESSAGE 149 ADDRESS	4296	
			UTILITY MESSAGE 150 ADDRESS	4297	
			UTILITY MESSAGE 151 ADDRESS	4298	
			UTILITY MESSAGE 152 ADDRESS	4299	
			UTILITY MESSAGE 153 ADDRESS	4300	
			UTILITY MESSAGE 154 ADDRESS	4301	
			UTILITY MESSAGE 155 ADDRESS	4302	
			UTILITY MESSAGE 156 ADDRESS	4303	
			UTILITY MESSAGE 157 ADDRESS	4304	
			UTILITY MESSAGE 158 ADDRESS	4305	
			UTILITY MESSAGE 159 ADDRESS	4306	
			UTILITY MESSAGE 160 ADDRESS	4307	
			UTILITY MESSAGE 161 ADDRESS	4308	
			UTILITY MESSAGE 162 ADDRESS	4309	
			UTILITY MESSAGE 163 ADDRESS	4310	
			UTILITY MESSAGE 164 ADDRESS	4311	
			UTILITY MESSAGE 165 ADDRESS	4312	
			UTILITY MESSAGE 166 ADDRESS	4313	
			UTILITY MESSAGE 167 ADDRESS	4314	
			UTILITY MESSAGE 168 ADDRESS	4315	
			UTILITY MESSAGE 169 ADDRESS	4316	
			UTILITY MESSAGE 170 ADDRESS	4317	
			UTILITY MESSAGE 171 ADDRESS	4318	
			UTILITY MESSAGE 172 ADDRESS	4319	
			UTILITY MESSAGE 173 ADDRESS	4320	
			UTILITY MESSAGE 174 ADDRESS	4321	
			UTILITY MESSAGE 175 ADDRESS	4322	
			UTILITY MESSAGE 176 ADDRESS	4323	
			UTILITY MESSAGE 177 ADDRESS	4324	
			UTILITY MESSAGE 178 ADDRESS	4325	
			UTILITY MESSAGE 179 ADDRESS	4326	
			UTILITY MESSAGE 180 ADDRESS	4327	
			UTILITY MESSAGE 181 ADDRESS	4328	
			UTILITY MESSAGE 182 ADDRESS	4329	
			UTILITY MESSAGE 183 ADDRESS	4330	
			UTILITY MESSAGE 184 ADDRESS	4331	
			UTILITY MESSAGE 185 ADDRESS	4332	
			UTILITY MESSAGE 186 ADDRESS	4333	
			UTILITY MESSAGE 187 ADDRESS	4334	
			UTILITY MESSAGE 188 ADDRESS	4335	
			UTILITY MESSAGE 189 ADDRESS	4336	
			UTILITY MESSAGE 190 ADDRESS	4337	
			UTILITY MESSAGE 191 ADDRESS	4338	
			UTILITY MESSAGE 192 ADDRESS	4339	
			UTILITY MESSAGE 193 ADDRESS	4340	
			UTILITY MESSAGE 194 ADDRESS	4341	
			UTILITY MESSAGE 195 ADDRESS	4342	
			UTILITY MESSAGE 196 ADDRESS	4343	
			UTILITY MESSAGE 197 ADDRESS	4344	
			UTILITY MESSAGE 198 ADDRESS	4345	
			UTILITY MESSAGE 199 ADDRESS	4346	
			UTILITY MESSAGE 200 ADDRESS	4347	
			UTILITY MESSAGE 201 ADDRESS	4348	
			UTILITY MESSAGE 202 ADDRESS	4349	
			UTILITY MESSAGE 203 ADDRESS	4350	
			UTILITY MESSAGE 204 ADDRESS	4351	
			UTILITY MESSAGE 205 ADDRESS	4352	
			UTILITY MESSAGE 206 ADDRESS	4353	
			UTILITY MESSAGE 207 ADDRESS	4354	
			UTILITY MESSAGE 208 ADDRESS	4355	
			UTILITY MESSAGE 209 ADDRESS	4356	
			UTILITY MESSAGE 210 ADDRESS	4357	
			UTILITY MESSAGE 211 ADDRESS	4358	
			UTILITY MESSAGE 212 ADDRESS	4359	
			UTILITY MESSAGE 213 ADDRESS	4360	
			UTILITY MESSAGE 214 ADDRESS	4361	
			UTILITY MESSAGE 215 ADDRESS	4362	
			UTILITY MESSAGE 216 ADDRESS	4363	
			UTILITY MESSAGE 217 ADDRESS	4364	
			UTILITY MESSAGE 218 ADDRESS	4365	
			UTILITY MESSAGE 219 ADDRESS	4366	
			UTILITY MESSAGE 220 ADDRESS	4367	
			UTILITY MESSAGE 221 ADDRESS	4368	
			UTILITY MESSAGE 222 ADDRESS	4369	
			UTILITY MESSAGE 223 ADDRESS	4370	
			UTILITY MESSAGE 224 ADDRESS	4371	
			UTILITY MESSAGE 225 ADDRESS	4372	
			UTILITY MESSAGE 226 ADDRESS	4373	
			UTILITY MESSAGE 227 ADDRESS	4374	
			UTILITY MESSAGE 228 ADDRESS	4375	
			UTILITY MESSAGE 229 ADDRESS	4376	
			UTILITY MESSAGE 230 ADDRESS	4377	
			UTILITY MESSAGE 231 ADDRESS	4378	
			UTILITY MESSAGE 232 ADDRESS	4379	
			UTILITY MESSAGE 233 ADDRESS	4380	
			UTILITY MESSAGE 234 ADDRESS	4381	

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTHTZ 33RUC	MLJ	LOC	OBJ
		=2111 ;					
		=2112 ;	MEMORY SPACE MODIFIER OPTION RESPONSE STRINGS				
		=2113 ;					
		=2114 DFRMEM:					
015F	73	=2115	DB 01110011B, 11010000B, 01001110B, "PR "				
0160	D0	=					
		=2116 DDAMEM:					
0161	5E	=2117	DB 01011110B, 11110111B, "DL "				
0162	F7	=	" 00 ", 00011011, 01011110B				
		=2118 DRM:					
0163	50	=2119	DB 01010000B, 10111101B, "RG "				
0164	BD	=	" 11 ", 00001110, 00000110B, 01000111B				
		=2120 DFRBKK:					
0165	73	=2121	DB 01110011B, 11111100B, "PB "				
0166	FC	=					
		=2122 DDABKK:	" 12 ", 00001111, 01010110B, 00001110B				
0167	5E	=2123	DB 01011110B, 11111100B, "DE "				
0168	FC	=					
		=2124 DINTRG:					
0169	76	=2125	DB 01110110B, 11010000B, 01100111B, "HR "				
016A	D0	=					
		=2126 ;					
		=2127 ;	RESPONSE MESSAGES FOR GO CONDITION MODIFIERS.				
		=2128 ;	" 10 ", 00001110, 00010101B, 01111101B				
		=2129 DNOBKK:					
016B	54	=2130	DB 01010100B, 11111100C, "ND "				
016C	FC	=					
		=2131 DWORK:	" 07 ", 00011111, 00001110B, 00011110B				
016D	7C	=2132	DB 01111100B, 11010000B, "BR "				
016E	D0	=					
		=2133 DSS:					
016F	6D	=2134	DB 01101101B, 01101101B, 11111000B, "SST "				
0170	6D	=					
0171	F8	=					
		=2135 DPA:					
0172	77	=2136	DB 01110111B, 01111100B, 11010000B, "ABR "				
0173	7C	=					
0174	D0	=					
		=2137 DTR:					
0175	77	=2138	DB 01110111B, 01101101B, 11111000B, "AST "				
0176	6D	=					
0177	F8	=					
		=2139 ;					
		=2140	SIZECHK				
0078		=2143+	SIZE SET 120				
		=2144+;					
		=2145+;	*****				
		=2154	\$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	LINE	LOC
		=2155	CODEBLK 45	000000	000000	
00C0		=2160+	ORG 192	000000	000000	0000
		=2164	INPADR INPUT DATA INTO TWO-BYTE PARAMETER BUFFER INDICATED BY R0.	000000	000000	
		=2165 ;	RECEIVE NUMERIC KEYS FROM KEYBOARD UNTIL ' ' OR ' '.	000000	000000	
		=2166 ;	SHIFT INTO ADDRESS BUFFER;	000000	000000	
		=2167 ;	RE-WRITE DISPLAY.	000000	000000	
		=2168 ;	IF NUMBER OF CONSTANTS NEEDED IS ZERO, NO NEW PARAMETERS ARE ALLOWED.	000000	000000	
		=2169 ;		000000	000000	
00C0 97		=2170	INPADR: CLR C	000000	000000	0000
00C1 A7		=2171	CPL C	000000	000000	0000
		=2172	MNOV A, NUMCON	000000	000000	0000
00C2 B938		=2181+	MOV R1, NUMCON	000000	000000	0000
00C4 F1		=2182+	MOV A, R01	000000	000000	0000
00C5 C6D7		=2186	JZ ELSIF1	000000	000000	0000
00C7 FB		=2187	INPAD1: MOV A, KEY	000000	000000	0000
00C8 92D7		=2188	JB4 ELSIF1	000000	000000	0000
00CA 20		=2189	XCH A, R00	000000	000000	0000
00CB 47		=2190	SWAP A	000000	000000	0000
00CC 20		=2191	XCH A, R00	000000	000000	0000
00CD 30		=2192	XCHD A, R00	000000	000000	0000
00CE 18		=2193	INC R0	000000	000000	0000
00CF 30		=2194	XCHD A, R00	000000	000000	0000
00D0 3478		=2195	CALL UPDADR	000000	000000	0000
00D2 14EC		=2196	CALL INPKEY	000000	000000	0000
00D4 97		=2197	CLR C	000000	000000	0000
00D5 04C7		=2198	JMP INPAD1	000000	000000	0000
		=2199 ;		000000	000000	0000
		=2200	ELSIF1 IF KEY=' ' OR ' ' THEN RETURN.	000000	000000	0000
		=2201 ;		000000	000000	0000
00D7 FB		=2202	ELSIF1: MOV A, KEY	000000	000000	0000
00D8 D312		=2203	XRL A, #KEYNXT	000000	000000	0000
00DA C6E5		=2204	JZ ELSIF2	000000	000000	0000
00DC FB		=2205	MOV A, KEY	000000	000000	0000
00DD D313		=2206	XRL A, #KEYEND	000000	000000	0000
00DF C6E5		=2207	JZ ELSIF2	000000	000000	0000
		=2208 ;		000000	000000	0000
		=2209 ;	ELSE GOTO PERROR.	000000	000000	0000
		=2210 ;		000000	000000	0000
00E1 B802		=2211	MOV LDATA, #2	000000	000000	0000
00E3 249A		=2212	JMP PERROR	000000	000000	0000
00E5 B846		=2213	ELSIF2: MOV R0, #SEGMAP	000000	000000	0000
00E7 B903		=2214	MOV R1, #3	000000	000000	0000
00E9 B4F5		=2215	CALL DELANK	000000	000000	0000
00EB 83		=2216	RET	000000	000000	0000
		=2217	SIZECHK	000000	000000	0000
002C		=2220+	SIZE SET 44	000000	000000	0000
		=2221+;		000000	000000	0000
		=2222+;	*****	000000	000000	0000
		=2231	\$EJECT	000000	000000	0000

	=2232	CODEBK 35		
0178	=2242+	ORG 376		
	=2246 ;	UPADR UPDATE ADDRESS FIELD		
	=2247 ;	(LAST THREE CHARACTERS OF DISPLAY) WITH ADDRESS BUFFER		
	=2248 UPADR:	MOV NEXTPL, PLUS3		
0178 D93A	=2259+	MOV R1, #NEXTPL		
017A D103	=2260+	MOV R1, #PLUS3		
	=2264 ;	WRITE ADDR INTO NEXT THREE BUFFER LOCATIONS.		
017C F0	=2265 UPADR1:	MOV A, R0		
017D C0	=2266	DEC R0		
017E 530F	=2267	ANL A, #0FH		
0180 960E	=2268	JNZ DSPHI		
0182 D408	=2269	CALL WDISP		
0184 F0	=2270	MOV A, R0		
0185 47	=2271	SWAP A		
0186 530F	=2272	ANL A, #0FH		
0188 9692	=2273	JNZ DSPM1		
018A D408	=2274	CALL WDISP		
018C 2494	=2275	JMP DSPLO		
018E D403	=2276 DSPHI:	CALL DSPACC		
0190 F0	=2277 DSPMID:	MOV A, R0		
0191 47	=2278	SWAP A		
0192 D403	=2279 DSPM1:	CALL DSPACC		
0194 F0	=2280 DSPLO:	MOV A, R0		
0195 D403	=2281	CALL DSPACC		
0197 83	=2282	RET		
	=2283	SIZECHK		
0020	=2286+	SIZE SET 32		
	=2287+			
	=2288+;	*****		
	=2297	#EJECT		

LOC	OBJ	LINE	SOURCE STATEMENT	THWENTATZ	35R002	3MLJ	LOB	30J
		=2298	CODEBLK 35	00117H	1100	1100	0150	0150
0198		=2300+	ORG 400	0110H	0100	0100	0150	0150
		=2312 ;	ERROR: REPEAT					
		=2313 ;	OUTPUT_MESSAGE(PERROR,PROMPT)	0100H	1100	1100	0150	0150
		=2314 ;	OUTPUT(LDATA)	0110H	0100	0100	0150	0150
		=2315 ;	CALL INPUT_BYTE(KEY)					
		=2316 ;	UNTIL KEY='CLEAR/KEYVIOUS'	01	0100	0100	0150	0150
0198 B804		=2317 ERROR: MOV	LDATA, #4	01	0100	0100	0150	0150
019A BF02		=2318 ERROR: MOV	XPCODE, #2	0110H	1100	1100	0150	0150
019C 7401		=2319	CALL XPTST	0110H	0100	0100	0150	0150
019E 27		=2320	CLR A					
019F D7		=2321	MOV PSW, A	0110H	0100	0100	0150	0150
01A0 FB		=2322	MOV A, KEY					
01A1 D317		=2323	XRL A, #KEYCLR	0110H	1100	1100	0150	0150
01A3 C606		=2324	JZ ERROR2	0110H	0100	0100	0150	0150
01A5 27		=2325	CLR A					
01A6 3400		=2326	CALL OUTUTL	0110H	1100	1100	0150	0150
01A8 FA		=2327	MOV A, LDATA					
01A9 D403		=2328	CALL DSPACC	0110H	1100	1100	0150	0150
		=2329	MNOV KBDUF, NEG1					
01AB B93B		=2340+	MOV R1, #KBDUF	0110H	1100	1100	0150	0150
01AD B11F		=2341+	MOV R1, #NEG1					
01AF 14EC		=2345	CALL INPKEY	0110H	1100	1100	0150	0150
01B1 FB		=2346	MOV A, KEY					
01B2 D313		=2347	XRL A, #KEYEND	0110H	1100	1100	0150	0150
01B4 9698		=2348	JNZ ERROR2	0110H	0100	0100	0150	0150
01B6 0429		=2349 ERROR2: JMP	MAIN	0110H	0100	0100	0150	0150
		=2350	SIZECHK					
0020		=2353+	SIZE SET 32					
		=2354+						
		=2355+	*****					
		=2364 ;						
		=2365	CODEBLK 80	00117H	1100	1100	0150	0150
0200		=2380+	ORG 512	0110H	0100	0100	0150	0150
		=2384 ;	IMPLEM IMPLEMENT COMMAND					
0200 2306		=2385 IMPLEM: MOV	A, #LOW(JMPTBL)	0110H	1100	1100	0150	0150
		=2386	RADD A, BCODE	0110H	1100	1100	0150	0150
0202 B936		=2392+	MOV R1, #BCODE	0110H	1100	1100	0150	0150
0204 61		=2393+	RDD A, R1	0110H	1100	1100	0150	0150
0205 B3		=2397	JMPP R1	0110H	1100	1100	0150	0150
		=2398 ;						
		=2399 JMPTBL:						
0206 0F		=2400	DB LOW(JTOMOD)	0110H	1100	1100	0150	0150
0207 20		=2401	DB LOW(JTODG)	0110H	1100	1100	0150	0150
0208 22		=2402	DB LOW(JTOFIL)	0110H	1100	1100	0150	0150
0209 1A		=2403	DB LOW(JTOLST)	0110H	1100	1100	0150	0150
020A 11		=2404	DB LOW(JTOREC)	0110H	1100	1100	0150	0150
020B 16		=2405	DB LOW(JTOREL)	0110H	1100	1100	0150	0150
020C 2C		=2406	DB LOW(COMSDR)	0110H	1100	1100	0150	0150
020D 28		=2407	DB LOW(COMCBR)	0110H	1100	1100	0150	0150
020E 26		=2408	DB LOW(JGORES)	0110H	1100	1100	0150	0150
		=2409 ;						
020F 444F		=2410 JTOMOD: JMP	EXAMIN	0110H	1100	1100	0150	0150
		=2411 ;						
0211 85		=2412 JTODG: CLR	F0 ; F0=0 ==> HEX FORMAT DATA DUMP	0110H	1100	1100	0150	0150

LOC	OBJ	LINE	SOURCE STATEMENT	DISASSEMBLY	LINE	LOC
0212	B472	=2413	CALL HFILED	CALL HFILED	0212	B472
0214	0429	=2414	JMP MAIN	JMP MAIN	0214	0429
		=2415 ;				
0216	5497	=2416	JTOREL CALL HRECIN	JTOREL CALL HRECIN	0216	5497
0218	0429	=2417	JMP MAIN	JMP MAIN	0218	0429
		=2418 ;				
021A	85	=2419	JTOLST: CLR F0	JTOLST: CLR F0	021A	85
021B	95	=2420	CPL F0	CPL F0	021B	95
021C	B472	=2421	CALL HFILED	CALL HFILED	021C	B472
021E	0429	=2422	JMP MAIN	JMP MAIN	021E	0429
		=2423 ;				
0220	8400	=2424	JTOGO: JMP EPRUN	JTOGO: JMP EPRUN	0220	8400
		=2425 ;				
0222	54E5	=2426	JTOFIL: CALL COMFIL	JTOFIL: CALL COMFIL	0222	54E5
0224	0429	=2427	JMP MAIN	JMP MAIN	0224	0429
		=2428 ;				
0226	8461	=2429	JGORES: JMP COMGOR	JGORES: JMP COMGOR	0226	8461
		=2430 ;				
		=2431 ; COMCBR COMMAND TO CLEAR BREAKPOINTS				
0228	B800	=2432	COMCBR: MOV LDATA, #0	COMCBR: MOV LDATA, #0	0228	B800
022A	442E	=2433	JMP BRKFIL	JMP BRKFIL	022A	442E
		=2434 ;				
		=2435 ; COMCBR COMMAND TO SET BREAKPOINTS				
022C	B801	=2436	COMCBR: MOV LDATA, #1	COMCBR: MOV LDATA, #1	022C	B801
022E	2304	=2437	BRKFIL: MOV A, #4	BRKFIL: MOV A, #4	022E	2304
		=2438	MADD TYPE, A	MADD TYPE, A		
0230	B937	=2448+	MOV R1, #TYPE	MOV R1, #TYPE	0230	B937
0232	61	=2449+	ADD A, @R1	ADD A, @R1	0232	61
0233	R1	=2455+	MOV @R1, A	MOV @R1, A	0233	R1
0234	F400	=2459	BRKNXT: CALL LSTORE	BRKNXT: CALL LSTORE	0234	F400
0236	FB	=2460	MOV A, KEY	MOV A, KEY	0236	FB
0237	D313	=2461	XRL A, #KEYEND	XRL A, #KEYEND	0237	D313
0239	C64D	=2462	JZ BRKEND	JZ BRKEND	0239	C64D
023B	14EC	=2463	CALL IMPKEY	CALL IMPKEY	023B	14EC
		=2464	MMOV NUMCON, PLUS1	MMOV NUMCON, PLUS1		
023D	D938	=2475+	MOV R1, #NUMCON	MOV R1, #NUMCON	023D	D938
023F	D101	=2476+	MOV @R1, #PLUS1	MOV @R1, #PLUS1	023F	D101
0241	B838	=2480	MOV R0, #SMALO	MOV R0, #SMALO	0241	B838
0243	B000	=2481	MOV @R0, #0	MOV @R0, #0	0243	B000
		=2482	MMOV SMALH, ZERO	MMOV SMALH, ZERO		
0245	D931	=2493+	MOV R1, #SMALH	MOV R1, #SMALH	0245	D931
0247	B100	=2494+	MOV @R1, #ZERO	MOV @R1, #ZERO	0247	B100
0249	14C0	=2498	CALL IMPADR	CALL IMPADR	0249	14C0
024B	E634	=2499	JNC BRKNXT	JNC BRKNXT	024B	E634
024D	0429	=2500	BRKEND: JMP MAIN	BRKEND: JMP MAIN	024D	0429
		=2501	SIZECHK	SIZECHK		
004F		=2504+	SIZE SET 79	SIZE SET 79	004F	
		=2505+				
		=2506+; *****				
		=2515	\$EJECT	\$EJECT		

LOC	OBJ	LINE	SOURCE STATEMENT	DISPATCH	STATEMENT	LINE	LOC
		=2617 ;					
027B	D313	=2618 EXAM1: XRL	A, # (KEYEND)				
027D	9681	=2619	JNZ EXAM2				
027F	0429	=2620	JMP MAIN				
		=2621 ;					
0281	FB	=2622 EXAM2: MOV	A, KEY				
0282	D312	=2623	XRL A, #KEYNXT				
0284	968A	=2624	JNZ EXAM3				
0286	34F2	=2625	CALL INCSMA				
0288	444F	=2626	JMP EXAMIN				
028A	FB	=2627 EXAM3: MOV	A, KEY				
028B	D317	=2628	XRL A, #KEYCLR				
028D	9693	=2629	JNZ EXAM4				
028F	54F4	=2630	CALL DECSMA				
0291	444F	=2631	JMP EXAMIN				
0293	B103	=2632 EXAM4: MOV	LDATA, #03H				
0295	249A	=2633	JMP PERROR				
		=2634	SIZECHK				
0048		=2637+ SIZE	SET 72				
		=2638+;					
		=2639+; *****					
		=2648 ;					
		=2649	CODEBLK 4				
00EC		=2654+	OKG 236				
00EC	D4C2	=2658 INPKLY: CALL	KBDIN ; RETURNS KEY DEPRESSION IN A				
00EE	AD	=2659	MOV KEY, A				
00EF	83	=2660	RET				
		=2661	SIZECHK				
0004		=2664+ SIZE	SET 4				
		=2665+;					
		=2666+; *****					
		=2675 \$EJECT					

LOC	OBJ	LINE	SOURCE STATEMENT	TARGET1	TARGET2	STATUS	100	300
		2676 \$	INCLUDE(:F0:GOCOMS.MOD)					
		=2677	CODEBLK 210					
0400		=2697+	ORG 1024					
		=2701	EPRUN RUN EMULATION MODE.					
		=2702 ;	RELOAD EP WITH SYSTEM STATUS AND RELEASE					
		=2703 ;	SEQUENCE IS AS FOLLOWS:					
		=2704 ;	IF COMMAND WAS TERMINATED BY THE 'NEXT' KEY:					
		=2705 ;	STORE SMA INTO EP PC;					
		=2706 ;	STORE EP PC INTO TOP-OF-STACK (RELATIVE TO EP PSW);					
		=2707 ;	PASS EP R0;					
		=2708 ;	PASS EP PSW;					
		=2709 ;	PASS EP TIMER;					
		=2710 ;	PASS EP ACCUMULATOR;					
		=2711 ;						
0400	2302	=2712 EPRUN:	MOV R, #2					
0402	3400	=2713	CALL OUTUTL					
		=2714	MMOV R, NUMCON					
0404	B938	=2723+	MOV R1, #NUMCON					
0406	F1	=2724+	MOV R, @R1					
0407	9615	=2728	JNZ EPCONT					
		=2729	MMOV EPPCLO, SMAILO					
0409	B930	=2745+	MOV R1, #SMAILO					
040B	F1	=2746+	MOV R, @R1					
040C	B924	=2752+	MOV R1, #EPPCLO					
040E	R1	=2753+	MOV @R1, R					
		=2756	MMOV EPPCHI, SMAHI					
040F	B931	=2772+	MOV R1, #SMAHI					
0411	F1	=2773+	MOV R, @R1					
0412	B925	=2779+	MOV R1, #EPPCHI					
0414	R1	=2780+	MOV @R1, R					
0415	FB	=2783	EPCONT: MOV R, KEY					
0416	D312	=2784	XRL R, #KEYNXT					
0418	C61F	=2785	JZ EPCON1					
041A	2301	=2786	MOV R, #01H ; STACK ONE LEVEL DEEP TO HOLD USER STARTING ADDRESS					
		=2787	MMOV EPPSW, R					
041C	B921	=2800+	MOV R1, #EPPSW					
041E	R1	=2801+	MOV @R1, R					
		=2805	EPCON1: MMOV LDATA, EPPCLO					
041F	B924	=2821+	MOV R1, #EPPCLO					
0421	F1	=2822+	MOV R, @R1					
0422	AA	=2835+	MOV LDATA, R					
		=2838	MMOV R, EPPSW					
0423	B921	=2847+	MOV R1, #EPPSW					
0425	F1	=2848+	MOV R, @R1					
0426	07	=2852	DEC R					
		=2853	ANL R, #07H					
0429	E7	=2854	RL R					
042A	0308	=2855	ADD R, #00H					
		=2856	MMOV SMAILO, R					
042C	B930	=2869+	MOV R1, #SMAILO					
042E	R1	=2870+	MOV @R1, R					
042F	F4C3	=2874	CALL EPSTOR					
		=2875	MINC SMAILO					
0431	B930	=2880+	MOV R1, #SMAILO					
0433	F1	=2881+	MOV @R1, R					

0434 17	=2885+	INC	A		
0435 A1	=2890+	MOV	0R1, A		
	=2893	MNOV	A, EPPSW		
0436 B921	=2902+	MOV	R1, #EPPSW		
0438 F1	=2903+	MOV	A, 0R1		
0439 53F0	=2907	ANL	A, #0F0H		
	=2908	MORL	A, EPPCHI		
043B B925	=2914+	MOV	R1, #EPPCHI		
043D 41	=2915+	ORL	A, 0R1		
043E 0A	=2919	MOV	LDATA, A		
043F F4C3	=2920	CALL	EPSTOR		
0441 B8D1	=2921 EPCNT:	MOV	R0, #LOW(0V2BAS+0VSIZE)		
0443 746A	=2922	CALL	OVLOAD		
	=2923	MNOV	A, EPR0		
0445 B923	=2932+	MOV	R1, #EPR0		
0447 F1	=2933+	MOV	A, 0R1		
0448 F4D0	=2937	CALL	EPPASS		
	=2938	MNOV	A, EPPSW		
044A B921	=2947+	MOV	R1, #EPPSW		
044C F1	=2948+	MOV	A, 0R1		
044D F4D0	=2952	CALL	EPPASS		
	=2953	MNOV	A, EPTIMR		
044F B922	=2962+	MOV	R1, #EPTIMR		
0451 F1	=2963+	MOV	A, 0R1		
0452 F4D0	=2967	CALL	EPPASS		
	=2968	MNOV	A, EPACC		
0454 B920	=2977+	MOV	R1, #EPACC		
0456 F1	=2978+	MOV	A, 0R1		
0457 F4D0	=2982	CALL	EPPASS		
0459 8903	=2983	ORL	P1, #00000011B		
045B F4D0	=2984	CALL	EPSTEP		
045D 745A	=2985	CALL	OVSMP		
045F 846B	=2986	JMP	CGO		
	=2987 ;				
	=2988 ;	COMGOR	GO FROM RESET COMMAND		
	=2989 ;		RESET PROCESSOR		
	=2990 ;		RELOAD LOW ORDER PROGRAM BYTES INTO PROGRAM MEMORY		
	=2991 ;				
0461 2302	=2992 COMGOR:	MOV	A, #2		
0463 3400	=2993	CALL	OUTUTL		
0465 8910	=2994	ORL	P1, #EPRSET		
0467 745A	=2995	CALL	OVSMP		
0469 99EF	=2996	ANL	P1, #(NOT EPRSET)		
	=2997 ;				
	=2998 ;				
	=2999 ;	CGO	SET UP BREAK LOGIC FOR APPROPRIATE BREAK CONDITIONS,		
	=3000 ;		DEPENDING ON CONTENTS OF 'TYPE'.		
	=3001 ;				
	=3002 CGO:	MNOV	A, TYPE		
046B B937	=3011+	MOV	R1, #TYPE		
046D F1	=3012+	MOV	A, 0R1		
046E 0371	=3016	ADD	A, #LOW GOTBL		
0470 B3	=3017	JMP	#1		
	=3018 ;				
0471 7C	=3019 GOTBL:	DE	LOW(CGONE)		

LOC	OBJ	LINE	SOURCE STATEMENT	DISASSEMBLY	HEX	LOC	OBJ
0472	76	=3020	DB LOW(CGOMB)	00000000	00000000		
0473	80	=3021	DB LOW(CGOS5)	00000000	00000000		
0474	76	=3022	DB LOW(CGOPAT)	00000000	00000000		
0475	80	=3023	DB LOW(CGOTRA)	00000000	00000000		
		=3024 ;					
		=3025 CGOPAT:					
0476	99FD	=3026 CGOMB: ANL	P1, #NOT 00000010C	00000000	00000000		
0478	8501	=3027	OKL P1, #00000001B	00000001	00000001		
047A	8482	=3028	JMP EPRUN4	00000000	00000000		
		=3029 ;					
047C	99FC	=3030 CGOMB: ANL	P1, #NOT 00000011B	00000001	00000001		
047E	8482	=3031	JMP EPRUN4	00000000	00000000		
		=3032 ;					
		=3033 CGOTRA:					
0480	8903	=3034 CGOS5: ORL	P1, #00000011B	00000001	00000001		
		=3035 ;					
		=3036 ; EPRUN4 SET UP CONTROL LOGIC TO RUN USER'S PROGRAM.					
		=3037 ;	RELEASE PROCESSOR TO RUN.				
		=3038 ;					
0482	8A20	=3039 EPRUN4: ORL	P2, #00100000B ; DISABLE EP LINK REFERENCES.	00100000	00100000		
0484	9A1F	=3040	ANL P2, #NOT 00010000B ; SET ALL REFERENCES TO RAM ARRAY.	00010000	00010000		
0486	99DF	=3041	ANL P1, #NOT MODOUT	00000000	00000000		
0488	F4F4	=3042	CALL EPREL	00000000	00000000		
		=3043 ;					
		=3044 ;	WAIT FOR KEYSTROKE INPUT OR HARDWARE BREAK TO OCCUR.				
		=3045 ;					
048A	F4AC	=3046 EPRUN1: CALL	10FPOL	00000000	00000000		
048C	F4AF	=3047	CALL KBOPOL	00000000	00000000		
048E	37	=3048	CPL A	00000000	00000000		
048F	F295	=3049	JB7 EPRUN3	00000000	00000000		
0491	8699	=3050	JNI EPRUN2	00000000	00000000		
0493	848A	=3051	JMP EPRUN4	00000000	00000000		
		=3052 ;					
		=3053 ; EPRUN3 A KEYSTROKE WAS DETECTED WHILE EP WAS RUNNING.					
		=3054 ;	BREAK EXECUTION.				
		=3055 ;	PROCESS KEYSTROKE.				
0495	D400	=3056 EPRUN3: CALL	STSAVE	00000000	00000000		
0497	84B3	=3057	JMP EPRUN5	00000000	00000000		
		=3058 ;					
		=3059 ; EPRUN2 AN ENABLED BREAK CONDITION OCCURRED.					
		=3060 ;	BREAK EMULATION MODE.				
		=3061 ;	CONTINUE ACCORDING TO GO COMMAND TYPE.				
0499	B400	=3062 EPRUN2: CALL	STSAVE	00000000	00000000		
		=3063	MOV A, TYPE	00000000	00000000		
049B	B937	=3072+	MOV R1, #TYPE	00000000	00000000		
049D	F1	=3073+	MOV A, @R1	00000000	00000000		
049E	03A1	=3077	ADD A, #LOW CNTTBL	00000000	00000000		
04A0	B3	=3078	JMPP @A	00000000	00000000		
		=3079 ;					
04A1	A6	=3080 CNTTBL: DB	LOW(BRKERR)	00000000	00000000		
04A2	BA	=3081	DB LOW(EPRUN6)	00000000	00000000		
04A3	BA	=3082	DB LOW(EPRUN6)	00000000	00000000		
04A4	AA	=3083	DB LOW(CNTTTRA)	00000000	00000000		
04A5	AA	=3084	DB LOW(CNTTTRA)	00000000	00000000		
		=3085 ;					

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTWO THREE	THIRTY	THIRTY
		=3086 ;	BRKERR BREAKPOINT LATCH WAS SET THOUGH BREAKPOINTS NOT ENABLED.		0580=	05 0780
		=3087 ;	DISPLAY HARDWARE ERROR MESSAGE.		0580=	05 0780
04A6	BA8B	=3088 BRKERR:	MOV LDATA, #00H		0580=	05 0780
04A8	249A	=3089	JMP PERORR		0580=	05 0780
		=3090 ;			0580=	05 0780
		=3091 CNTTRA:	MMOV R, DSPTIM		0580=	05 0780
04AA	B928	=3100+	MOV RL, #DSPTIM		0580=	05 0780
04AC	F1	=3101+	MOV R, @R1		0580=	05 0780
04AD	94F2	=3105	CALL DELAY		0580=	05 0780
04AF	F4AF	=3106	CALL KBDPOL		0580=	05 0780
04D1	F241	=3107	JB7 EPCNT ; B7 SET INDICATES NO KEYSTROKE		0580=	05 0780
		=3108 ;			0580=	05 0780
		=3109 ;	EPRUN5 INPUT(KEY),		0580=	05 0780
		=3110 ;	IF KEY=END GO TO PARSER.		0580=	05 0780
		=3111 ;	INPUT KEY,		0580=	05 0780
		=3112 ;	IF KEY<NEXT GO TO PARSER.		0580=	05 0780
		=3113 ;	CONTINUE IN SAME MODE.		0580=	05 0780
		=3114 ;			0580=	05 0780
04B3	14EC	=3115 EPRUN5:	CALL INPKEY		0580=	05 0780
04B5	FB	=3116	MOV R, KEY		0580=	05 0780
04B6	D313	=3117	XRL R, #KEYEND		0580=	05 0780
04B8	96C7	=3118	JNZ EPRET		0580=	05 0780
04DA	14EC	=3119 EPRUN6:	CALL INPKEY		0580=	05 0780
04BC	FB	=3120	MOV R, KEY		0580=	05 0780
04BD	D312	=3121	XRL R, #KEYNEXT		0580=	05 0780
04BF	96C7	=3122	JNZ LPRET		0580=	05 0780
04C1	2302	=3123	MOV R, #2		0580=	05 0780
04C3	3400	=3124	CALL OUTUTL		0580=	05 0780
04C5	8441	=3125	JMP EPCNT		0580=	05 0780
		=3126 ;			0580=	05 0780
		=3127 ;	EPRET EXECUTION MODE IS TO BE TERMINATED.		0580=	05 0780
		=3128 ;	JUMP INTO PARSER TO INTERPRET KEY ALREADY DETECTED.		0580=	05 0780
04C7	0433	=3129 EPRET:	JMP MAIN2		0580=	05 0780
		=3130 ;			0580=	05 0780
		=3131	SIZECHK		0580=	05 0780
00C9		=3134+ SIZE	SET 201		0580=	05 0780
		=3135+;			0580=	05 0780
		=3136+; *****			0580=	05 0780
		=3145 \$EJECT			0580=	05 0780

0541 E7	=3299	RL	A		
0542 0300	=3300	ADD	A, #00H		
	=3301	MMOV	SMALO, A		
0544 B930	=3314+	MOV	RL, #SMALO		
0546 R1	=3315+	MOV	ORL, A		
0547 F487	=3319	CALL	EPFET		
0549 03FE	=3320	ADD	A, #2		
054B AA	=3321	MOV	LDATA, A		
	=3322	MMOV	EPPCLO, A		
054C B924	=3335+	MOV	RL, #EPPCLO		
054E R1	=3336+	MOV	ORL, A		
054F F4C3	=3340	CALL	EPSTOR		
0551 B930	=3341	MOV	RL, #SMALO		
0553 11	=3342	INC	ORL		
0554 F487	=3343	CALL	EPFET		
0556 AA	=3344	MOV	LDATA, A		
0557 53F0	=3345	ANL	A, #11110000B		
0559 2A	=3346	XCH	A, LDATA		
055A 13FF	=3347	ADDC	A, #1		
055C 530F	=3348	ANL	A, #00001111B		
	=3349	MMOV	EPPOCHI, A		
055E B925	=3362+	MOV	RL, #EPPCHI		
0560 R1	=3363+	MOV	ORL, A		
0561 4A	=3367	ORL	A, LDATA		
0562 AA	=3368	MOV	LDATA, A		
0563 F4C3	=3369	CALL	EPSTOR		
0565 0825	=3370	MOV	R0, #EPPCHI		
0567 347C	=3371	CALL	UPDAD1		
0569 2340	=3372	MOV	A, #01000000B ; " FOR DISPLAY		
056B D408	=3373	CALL	WDISP		
056D B820	=3374	MOV	R0, #EPACC		
056F 3490	=3375	CALL	DSPMID		
0571 03	=3376	RET			
	=3377	SIZECHK			
0072	=3380+ SIZE SET 114				
	=3381+;				
	=3382+; *****				
	=3391 #EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTIME	LINE	LOC
		3392 \$	INCLUDE(=F0:HFILE.MOD)			
0000		=3393	CHARCR EQU 0DH ; <CR>			
000A		=3394	CHARLF EQU 0AH ; <LF>			
001A		=3395	CNTRLZ EQU 1AH ; CONTROL-Z			
		=3396	;			
		=3397	CODEBLK 80			
0297		=3412+	ORG 663			
		=3416	IRECIN HEXFILE RECORD INPUT ROUTINE			
0297 34CD		=3417	IRECIN: CALL CHARIN			
0299 D31A		=3418	XRL A, #CNTRLZ			
029B C6E0		=3419	JZ DONE			
029D D31A		=3420	XRL A, #CNTRLZ			
029F D33A		=3421	XRL A, #'(')			
02A1 9697		=3422	JNZ IRECIN			
		=3423	MMOV CHKSUM, ZERO			
02A3 D000		=3428+	MOV CHKSUM, #ZERO			
02A5 14F0		=3432	CALL BYTEIN			
		=3433	MMOV BUF CNT, A			
02A7 B941		=3446+	MOV R1, #BUF CNT			
02A9 A1		=3447+	MOV @R1, A			
02AA 14F0		=3451	CALL BYTEIN			
		=3452	MMOV SMARI, A			
02AC B931		=3465+	MOV R1, #SMARI			
02AE A1		=3466+	MOV @R1, A			
02AF 14F0		=3470	CALL BYTEIN			
		=3471	MMOV SMALO, A			
02B1 B930		=3484+	MOV R1, #SMALO			
02B3 A1		=3485+	MOV @R1, A			
02B4 14F0		=3489	CALL BYTEIN			
		=3490	MMOV RECTYP, A			
02B6 B942		=3503+	MOV R1, #RECTYP			
02B8 A1		=3504+	MOV @R1, A			
		=3508	;			
		=3509	HDATIN HEX DATA BYTE IN			
		=3510	HDATIN: MMOV A, BUF CNT			
02B9 B941		=3519+	MOV R1, #BUF CNT			
02BB F1		=3520+	MOV A, @R1			
02BC C6CC		=3524	JZ RECDON			
02BE 14F0		=3525	CALL BYTEIN			
02C0 AA		=3526	MOV LDATA, A			
02C1 F400		=3527	CALL LSTORE			
02C3 34F2		=3528	CALL INCSMA			
		=3529	MDEC BUF CNT			
02C5 B941		=3534+	MOV R1, #BUF CNT			
02C7 F1		=3535+	MOV A, @R1			
02C8 07		=3539+	DEC A			
02C9 A1		=3544+	MOV @R1, A			
02CA 44B9		=3547	JMP HDATIN			
		=3548	;			
02CC 34CD		=3549	RECDON: CALL CHARIN			
02CE D331		=3550	XRL A, #'('			
02D0 C6DB		=3551	JZ CKSMOK			
02D2 D33F		=3552	XRL A, #'('			
02D4 34BA		=3553	CALL NIBIN2			
02D6 14F2		=3554	CALL BYTEI1			

LOC	OBJ	LINE	SOURCE STATEMENT	INFORMATION	DISC	LOC	OBJ
		=3555		(RESULT FOR NON-?? CHARACTERS IS AS IF			
		=3556		BYTEIN WAS CALLED)			
		=3557	MMOV A,CHKSUM				
02D8	FD	=3573+	MOV A,CHKSUM				
02D9	9GE1	=3577	JNZ CHKERR				
		=3578	CKSMOK:MMOV A,RECTYP				
02D8	B942	=3587+	MOV R1,#RECTYP				
02D0	F1	=3588+	MOV A,R1				
02DE	C697	=3592	JZ HRECIN				
		=3593 ;					
		=3594 ;	DONE HEX FILE CORRECTLY RECEIVED				
02E0	83	=3595	DONE: RET				
		=3596 ;					
		=3597 ;	CHKERR CHECKSUM ERROR IN INPUT RECORD DETECTED				
02E1	BA0C	=3598	CHKERR: MOV LDATA,#0CH				
02E3	249A	=3599	JMP FLRORR				
		=3600	SIZECHK				
004E		=3603+	SIZE SET 78				
		=3604+					
		=3605+;	*****				
		=3614 ;					
		=3615	CODEBLK 12				
00F0		=3620+	ORG 240				
		=3624 ;	BYTEIN BYTE INPUT SUBROUTINE				
		=3625 ;	RECEIVES TWO HEXIDECIMAL CHARACTERS FROM THE TAPE INPUT DEVICE				
		=3626 ;	AND ASSEMBLES THEM INTO A SINGLE BYTE OF DATA				
00F0	34B8	=3627	BYTEIN: CALL NIBIN				
00F2	47	=3628	BYTE1: SWAP A				
00F3	AA	=3629	MOV LDATA,A				
00F4	34B8	=3630	CALL NIBIN				
		=3631	MORL LDATA,A				
00F6	4A	=3640+	ORL A,LDATA				
00F7	AA	=3660+	MOV LDATA,A				
00F8	6D	=3664	ADD A,CHKSUM				
00F9	AD	=3665	MOV CHKSUM,A				
00FA	FA	=3666	MOV A,LDATA				
00FB	83	=3667	RET				
		=3668	SIZECHK				
006C		=3671+	SIZE SET 12				
		=3672+;					
		=3673+;	*****				
		=3682 ;					
		=3683	CODEBLK 25				
01B8		=3693+	ORG 440				
		=3697 ;	NIBIN RECEIVES A HEXIDECIMAL CHARACTER AND PRODUCES A MASKED FOUR BIT VALUE				
		=3698 ;	NOTE- ERROR CHECKING DONE TO VERIFY HEXIDECIMAL VALIDITY				
01B8	34CD	=3699	NIBIN: CALL CHARIN				
01BA	03C6	=3700	NIBIN2: ADD A,#-3AH				
		=3701					
01BC	E6C2	=3702	JNC NIBI3				
01BE	03F9	=3703	ADD A,#-7				
01C0	E6C3	=3704	JNC ASCLR				
		=3705 ;					
		=3706 ;	ACC=0F01-05H FOR CHARACTERS '0'-'F'				
		=3707 ;					

LOC	OBJ	LINE	SOURCE STATEMENT	THIRSTAT2 338AC2	3111	130 301
01C2	03FA	=3708	NIBI3: ADD A, #6	; ACC=0F0H-0FFH FOR CHARACTERS '0'-'F'	030C=	
01C4	0310	=3709	ADD A, #10H	; ACC=00H-0FH FOR CHARACTERS '0'-'F';	+345C=	5728
		=3710		; OVERFLOW IF ABOVE IS TRUE.	+345C=	
01C6	E6C9	=3711	JNC ASCERR		+345C=	
01C8	83	=3712	RET		+345C=	
		=3713			+345C=	
		=3714	ASCERR ILLEGAL HEXIDECIMAL CHARACTER RECEIVED		+345C=	5728
01C9	8A0A	=3715	ASCERR: MOV LDATA, #0AH		+345C=	5728
01CB	249A	=3716	JMP PERROR		+345C=	5728
		=3717	SIZECHK		+345C=	5728
0015		=3720	SIZE SET 21		+345C=	5728
		=3721			+345C=	5728
		=3722	*****		+345C=	5728
		=3731			+345C=	5728
		=3732			+345C=	5728
		=3733	CODEBLK 5		+345C=	5728
01CD		=3743	ORG 461		+345C=	5728
		=3747	CHARIN CHARACTER INPUT ROUTINE.		+345C=	5728
		=3748	RECEIVES ONE ASCII CHARACTER FROM THE LOGICAL READER DEVICE.		+345C=	5728
01CD	D449	=3749	CHARIN: CALL CIN		+345C=	5728
01CF	537F	=3750	ANL A, #7FH		+345C=	5728
01D1	83	=3751	RET		+345C=	5728
		=3752	SIZECHK		+345C=	5728
0005		=3755	SIZE SET 5		+345C=	5728
		=3756			+345C=	5728
		=3757	*****		+345C=	5728
		=3766			+345C=	5728
		=3767			+345C=	5728
		=3768	\$EJECT		+345C=	5728

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOLIC ADDRESS	HEX	OBJ
		=3769	CODEBLK 100			
0572		=3794	ORG 1394			
		=3798	HFILCO HEX FILE OUTPUT SUBROUTINE			
		=3799 ;	WHEN CALLED WITH F0=0 OUTPUT IS STANDARD HEX FILE FORMAT.			
		=3800 ;	WHEN CALLED WITH F0=1 OUTPUT IS FORMATTED DATA DUMP TO CRT.			
		=3801	HFILCO: MOV MEMHI, SMARI			
0572 B931		=3817+	MOV R1, #SMARI			
0574 F1		=3818+	MOV A, @R1			
0575 B935		=3824+	MOV R1, #MEMHI			
0577 A1		=3825+	MOV @R1, A			
		=3826	MMOV MEMLO, SMALO			
0578 B930		=3844+	MOV R1, #SMALO			
057A F1		=3845+	MOV A, @R1			
057B B934		=3851+	MOV R1, #MEMLO			
057D A1		=3852+	MOV @R1, A			
		=3855	MMOV CHKSUM, ZERO			
057E B000		=3860+	MOV CHKSUM, #ZERO			
0580 B865		=3864	MOV R0, #HEXBUF			
		=3865 ;				
		=3866	LDBYTE: LOAD NEXT BYTE FROM MEMORY INTO HEX BUFFER			
0582 14FC		=3867	LDBYTE: CALL LFETCH			
0584 FA		=3868	MOV A, LDATA			
0585 A0		=3869	MOV @R0, A			
0586 18		=3870	INC R0			
0587 B4E2		=3871	CALL CPMFAS			
0589 E696		=3872	JNC ENDFIL			
058B 34F2		=3873	CALL INCSMA			
058D F8		=3874	MOV A, R0			
058E 0388		=3875	ADD A, #- (BUFLN+HEXBUF)			
0590 E682		=3876	JNC LDBYTE			
0592 D400		=3877	CALL HREC0			
0594 A472		=3878	JMP HFILCO			
		=3879 ;				
		=3880	ENDFIL: END HEX FILE TRANSMISSION:			
		=3881 ;	PRINT OUT BUFFER FOR LAST DATA RECORD			
		=3882 ;	PRINT OUT CANNED 'END-OF-FILE' RECORD			
		=3883 ;	RETURN.			
0596 D400		=3884	ENDFIL: CALL HREC0			
0598 D6A7		=3885	JF0 HFDONE			
059A 34D2		=3886	CALL TCRLF0			
059C B8AE		=3887	MOV R0, # (LOW EOFREC)			
059E F8		=3888	ENDF1: MOV A, R0			
059F A3		=3889	MOV A, @A			
05A0 C6A7		=3890	JZ HFDONE			
05A2 B4BD		=3891	CALL CHAR0			
05A4 18		=3892	INC R0			
05A5 A49E		=3893	JMP ENDF1			
05A7 34D2		=3894	HFDONE: CALL TCRLF0			
05A9 231A		=3895	MOV A, #CNTRLZ			
05AB B4BD		=3896	CALL CHAR0			
05AD 83		=3897	RET			
		=3898 ;				
		=3899	EOFREC: CHARACTER SKIPPING FOR CANNED END-OF-FILE RECORD FOR			
		=3900 ;	INTEL HEX FILE FORMAT STANDARD.			
05AE 203A3030		=3901	EOFREC: DC '00000001FF'			

LOC	OBJ	LINE	SOURCE STATEMENT	LOC	OBJ
05B2	30303030				
05B6	30314646				
05BA	00	=3902	DB 0 ;END OF STRING CODE BYTE		
		=3903	SIZECHK		
0049		=3906+	SIZE SET 73		
		=3907+			
		=3908+	*****		
		=3917 ;			
		=3918 ;			
		=3919	CODEBLK 90		
0600		=3949+	ORG 1536		
		=3953 ;	HRECO HEXDECIMAL RECORD OUTPUT SEQUENCE.		
		=3954 ;	HEX BUFFER ALREADY LOADED.		
0600	F8	=3955	HRECO: MOV A,R0		
0601	0398	=3956	ADD A,#-HEXBUF		
		=3957	MMOV BUFNT,A		
0603	B941	=3970+	MOV R1,#BUFNT		
0605	R1	=3971+	MOV @R1,A		
0606	3402	=3975	CALL TCRIFO		
0608	2320	=3976	MOV A,#'		
060A	B48D	=3977	CALL CHARO		
060C	D617	=3978	JF0 FDUMP1		
060E	233A	=3979	MOV A,#''		
0610	B48D	=3980	CALL CHARO		
		=3981	MMOV A,BUFNT		
0612	B941	=3990+	MOV R1,#BUFNT		
0614	F1	=3991+	MOV A,@R1		
0615	34DC	=3995	CALL BYTEO		
		=3996	FDUMP1: MMOV A,MEMHI		
0617	B935	=4005+	MOV R1,#MEMHI		
0619	F1	=4006+	MOV A,@R1		
061A	34DB	=4010	CALL BYTEO		
		=4011	MMOV A,MEMLO		
061C	B934	=4020+	MOV R1,#MEMLO		
061E	F1	=4021+	MOV A,@R1		
061F	34DB	=4025	CALL BYTEO		
0621	B628	=4026	JF0 FDUMP2		
0623	27	=4027	CLR A		
0624	34DB	=4028	CALL BYTEO		
0626	C42C	=4029	JMP DATO		
0628	233D	=4030	FDUMP2: MOV A,#''		
062A	B48D	=4031	CALL CHARO		
		=4032	DATO DATA OUTPUT		
062C	B865	=4033	DATO: MOV R0,#HEXBUF		
062E	B632	=4034	DATO1: JF0 FDUMP5		
0630	C436	=4035	JMP FDUMP3		
0632	2320	=4036	FDUMP5: MOV A,#'		
0634	B48D	=4037	CALL CHARO		
0636	F0	=4038	FDUMP3: MOV A,R0		
0637	34DB	=4039	CALL BYTEO		
0639	18	=4040	INC R0		
		=4041	MOJNZ BUFNT,DATO1		
063A	B941	=4046+	MOV R1,#BUFNT		
063C	F1	=4047+	MOV A,@R1		
063D	07	=4051+	DEC A		

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	LINE	LOC	OBJ
063E	RL	=4056+	MOV RCL A			063E	RL
063F	962E	=4060+	JNZ DAT01			063F	962E
		=4062 ;					
		=4063 ;	ENDREC END RECORD BEING TRANSMITTED				
0641	B648	=4064	ENDREC: JF0 FDUMP4			0641	B648
		=4065	MMOV A, CHKSUM				
0643	FD	=4081+	MOV A, CHKSUM			0643	FD
0644	37	=4085	CPL A			0644	37
0645	17	=4086	INC A			0645	17
0646	340B	=4087	CALL BYTE0			0646	340B
0648	83	=4088	FDUMP4: RET			0648	83
		=4089	SIZECHK				
0049		=4092+	SIZE SET 73			0049	
		=4093+;					
		=4094+;	*****				
		=4103 ;					
		=4104	CODEBLK 9				
01D2		=4114+	ORG 466			01D2	
		=4118 ;	TCRLF0 TAPE <CR><LF> OUTPUT				
01D2	230D	=4119	TCRLF0: MOV A, #CHARCR			01D2	230D
01D4	B4BD	=4120	CALL CHAR0			01D4	B4BD
01D6	230H	=4121	MOV A, #CHARLF			01D6	230H
01D8	B4BD	=4122	CALL CHAR0			01D8	B4BD
01DA	83	=4123	RET			01DA	83
		=4124	SIZECHK				
0009		=4127+	SIZE SET 9			0009	
		=4128+;					
		=4129+;	*****				
		=4138 ;					
		=4139	CODEBLK 11				
01D8		=4149+	ORG 475			01D8	
		=4153 ;	BYTE0 BYTE OUTPUT				
01D8	AR	=4154	BYTE0: MOV LDATA, A			01D8	AR
01DC	6D	=4155	ADD A, CHKSUM			01DC	6D
01DD	AD	=4156	MOV CHKSUM, A			01DD	AD
01DE	FA	=4157	MOV A, LDATA			01DE	FA
01DF	47	=4158	SWAP A			01DF	47
01E0	B4CB	=4159	CALL NIB0			01E0	B4CB
01E2	FA	=4160	MOV A, LDATA			01E2	FA
01E3	B4BB	=4161	CALL NIB0			01E3	B4BB
01E5	83	=4162	RET			01E5	83
		=4163	SIZECHK				
000B		=4166+	SIZE SET 11			000B	
		=4167+;					
		=4168+;	*****				
		=4177 ;					
		=4178	CODEBLK 12				
01E6		=4188+	ORG 48C			01E6	
		=4192 ;	HEXASC HEXDECIMAL NIBBLE TO ASCII CHARACTER CONVERSION				
01E6	530F	=4193	HEXASC: ANL A, #0FH			01E6	530F
01E8	03F6	=4194	ADD A, #(-10)			01E8	03F6
01EA	F6EF	=4195	JC HEXNIB			01EA	F6EF
01EC	033A	=4196	ADD A, #('0')			01EC	033A
01EE	83	=4197	RET			01EE	83
01EF	0341	=4198	HEXNIB: ADD A, #('A')			01EF	0341

```

01F1 83      =4199      RET
              =4200      SIZECHK
000C          =4203+ SIZE SET 12
              =4204+;
              =4205+; *****
              =4214 ;
              =4215 ;
              =4216 DECLARE BITS0.CONST1
000B          =4230 BITS0 EQU 11 ;DATA BITS PUT OUT (INCLUDING TWO STOP BITS)
              =4231 ;
              =4232      CODEBLK 30
04C9          =4252+      ORG 1225
              =4256 ;HBDLAY HALF-BIT TIME DELAY
              =4257 HBDLAY: MMOV H,HBITHI
04C9 B927     =4273+      MOV R1,HBITHI
04CB F1       =4274+      MOV A,R1
04CC B945     =4280+      MOV R1,#H
04CE A1       =4281+      MOV @R1,A
              =4284      MMOV R1,HBITLO
04CF B926     =4300+      MOV R1,HBITLO
04D1 F1       =4301+      MOV A,@R1
04D2 A9       =4314+      MOV R1,A
04D3 B4D7     =4317      J18 JMP HBD1
04D5 B900     =4318 HBD2: MOV R1,#0
04D7 E9D7     =4319 HBD1: DJNZ R1,HBD1
              =4320      HDJNZ H,HBD2
04D9 B945     =4325+      MOV R1,#H
04DB F1       =4326+      MOV A,@R1
04DC 07       =4330+      DEC A
04DD A1       =4335+      MMOV R1,@R1
04DE 96D5     =4339+      JNZ J18 HBD2
04E0 83       =4341      RET
              =4342      SIZECHK
0018          =4345+ SIZE SET 24
              =4346+;
              =4347+; *****
              =4356 ;
              =4357 $EJECT

```

1

LOC	OBJ	LINE	SOURCE STATEMENT	DISPATCH	DATE	160	301
065B	94C9	=4538	CALL HBDLAY				
065D	5662	=4539	JT1 C13 ;CHECK SID LINE LEVEL				
065F	97	=4540	CLR C ;DATA BIT IN CY				
0660	C465	=4541	JMP C14				
0662	97	=4542 C13:	CLR C				
0663	A7	=4543	CPL C				
0664	00	=4544	NOP ;EVEN OUT BRANCH EXECUTION TIMES				
0665	00	=4545 C14:	NOP				
0666	00	=4546	NOP				
0667	00	=4547	NOP				
		=4548	MRRC REGC				
0668	B944	=4553+	MOV R1, #REGC				
066A	F1	=4554+	MOV A, @R1				
066B	67	=4558+	RRC A				
066C	A1	=4563+	MOV @R1, A				
		=4566	MDJNZ B, C12				
066D	B943	=4571+	MOV R1, #B				
066F	F1	=4572+	MOV A, @R1				
0670	07	=4576+	DEC A				
0671	A1	=4581+	MOV @R1, A				
0672	9659	=4585+	JNZ C12				
		=4587	MNOV A, REGC				
0674	B944	=4596+	MOV R1, #REGC				
0676	F1	=4597+	MOV A, @R1				
0677	83	=4601	RET ; CHARACTER COMPLETE				
		=4602	SIZECHK				
062F		=4605+ SIZE SET 47					
		=4606+;					
		=4607+; *****					
		=4616 \$EJECT					

LOC	OBJ	LINE	SOURCE STATEMENT	INSTR	OP	DATA	PC	PSW	SP	BP	SI	DI	AX	CX	DX	BX	AX	CX	DX	BX
		4617 \$	INCLUDE(:F0:MEMREF.MOD)																	
		=4618	CODEBLK 15																	
02E5		=4633+	ORG 741																	
		=4637 ;	COMFIL COMMAND TO FILL ADDRESS SPACE BETWEEN SMA AND EMA WITH DATA																	
		=4638 ;	IN LOW BYTE OF MEM.																	
		=4639	COMFIL: MMOV LDATA, MEMLO																	
02E5 B934		=4655+	MOV R1, #MEMLO																	
02E7 F1		=4656+	MOV A, @R1																	
02E8 AA		=4669+	MOV LDATA, A																	
02E9 F400		=4672 LFILL:	CALL LSTORE																	
02EB B4E2		=4673	CALL CHPMAS																	
02ED E6F3		=4674	JNC LFILL1																	
02EF 34F2		=4675	CALL INCSMA																	
02F1 44E9		=4676	JMP LFILL																	
02F3 83		=4677 LFILL1:	RET																	
		=4678	SIZECHK																	
000F		=4681+ SIZE SET 15																		
		=4682+;																		
		=4683+;	*****																	
		=4692 ;																		
		=4693	CODEBLK 4																	
00FC		=4698+	ORG 252																	
		=4702 ;	LFETCH FETCHES CONTENTS OF LOGICAL MEMORY ADDRESS DETERMINED BY																	
		=4703 ;	<TYPE>, <SMHI>, & <SMLO> INTO <LDATA>.																	
00FC D478		=4704 LFETCH:	CALL AFETCH																	
00FE AA		=4705	MOV LDATA, A																	
00FF 83		=4706	RET																	
		=4707	SIZECHK																	
0004		=4710+ SIZE SET 4																		
		=4711+;																		
		=4712+;	*****																	
		=4721 ;																		
		=4722	CODEBLK 75																	
0678		=4752+	ORG 1656																	
		=4756 ;																		
		=4757 ;	AFETCH LOGICAL FETCH SUBROUTINE																	
		=4758 ;	FETCHS CONTENTS OF VARIOUS MEMORY SPACES TO ACC.																	
		=4759 AFETCH:	MMOV A, TYPE																	
0678 B937		=4768+	MOV R1, #TYPE																	
067A F1		=4769+	MOV A, @R1																	
067B 037E		=4773	ADD A, #LOW LFETBL																	
067D B3		=4774	JMPP @A																	
		=4775 ;																		
067E 04		=4776 LFETBL:	DB LOW LFEPH																	
067F 98		=4777	DB LOW LFEDM																	
0680 9C		=4778	DB LOW LFEREG																	
0681 A9		=4779	DB LOW LFEINT																	
0682 B1		=4780	DB LOW LFEBRK																	
0683 B1		=4781	DB LOW LFEBRK																	
		=4782 ;																		
		=4783 LFEPH:	MMOV A, SMHI																	
0684 B931		=4792+	MOV R1, #SMHI																	
0686 F1		=4793+	MOV A, @R1																	
0687 9698		=4797	JNZ LFEDM																	
		=4798	MMOV A, SMLO																	

LOC	OBJ	LINE	SOURCE STATEMENT	DISASSEMBLY	LOC	OBJ
0689	B930	=4887+	MOV R1, #SMALO	MOV R1, #SMALO	0689	B930
068B	F1	=4888+	MOV R1, #R1	MOV R1, #R1	068B	F1
068C	03E9	=4812	ADD R1, #OVSIZE	ADD R1, #OVSIZE	068C	03E9
068E	F698	=4813	JC LFEDM	JC LFEDM	068E	F698
		=4814	MMOV R1, #SMALO	MMOV R1, #SMALO		
0690	B930	=4823+	MOV R1, #SMALO	MOV R1, #SMALO	0690	B930
0692	F1	=4824+	MOV R1, #R1	MOV R1, #R1	0692	F1
0693	034E	=4820	ADD R1, #OVBUF	ADD R1, #OVBUF	0693	034E
0695	A9	=4829	MOV R1, A	MOV R1, A	0695	A9
0696	F1	=4830	MOV R1, #R1	MOV R1, #R1	0696	F1
0697	03	=4831	RET	RET	0697	03
0698	94E1	=4832	LFEDM: CALL LPSEL	LFEDM: CALL LPSEL	0698	94E1
069A	01	=4833	MOVX R1, #R1	MOVX R1, #R1	069A	01
069B	03	=4834	KLT	KLT	069B	03
		=4835 ;				
		=4836	LFEREG: MMOV R1, #SMALO	LFEREG: MMOV R1, #SMALO		
069C	B930	=4845+	MOV R1, #SMALO	MOV R1, #SMALO	069C	B930
069E	F1	=4846+	MOV R1, #R1	MOV R1, #R1	069E	F1
069F	537F	=4850	ANL R1, #01111111B ; CHECK IF LOW 7 BITS =0	ANL R1, #01111111B ; CHECK IF LOW 7 BITS =0	069F	537F
06A1	C6A5	=4851	JZ LFER0	JZ LFER0	06A1	C6A5
06A3	E4B7	=4852	JMP EPFET	JMP EPFET	06A3	E4B7
		=4853 ;				
		=4854	LFER0: MMOV R1, #EPR0	LFER0: MMOV R1, #EPR0		
06A5	B923	=4863+	MOV R1, #EPR0	MOV R1, #EPR0	06A5	B923
06A7	F1	=4864+	MOV R1, #R1	MOV R1, #R1	06A7	F1
06A8	03	=4868	RET	RET	06A8	03
		=4869 ;				
		=4870	LFEINT: MMOV R1, #SMALO	LFEINT: MMOV R1, #SMALO		
06A9	B930	=4879+	MOV R1, #SMALO	MOV R1, #SMALO	06A9	B930
06AB	F1	=4880+	MOV R1, #R1	MOV R1, #R1	06AB	F1
06AC	0320	=4884	ADD R1, #EPRACC	ADD R1, #EPRACC	06AC	0320
06AE	A9	=4885	MOV R1, A	MOV R1, A	06AE	A9
06AF	F1	=4886	MOV R1, #R1	MOV R1, #R1	06AF	F1
06B0	03	=4887	RET	RET	06B0	03
		=4888 ;				
		=4889	LFEBRK: LOGICAL FETCH OF BREAK-POINT DATA	LFEBRK: LOGICAL FETCH OF BREAK-POINT DATA		
06B1	94E1	=4890	LFEBRK: CALL LPSEL	LFEBRK: CALL LPSEL	06B1	94E1
06B3	99F7	=4891	ANL P1, #NOT 00001000B	ANL P1, #NOT 00001000B	06B3	99F7
06B5	8900	=4892	ORL P1, #00001000B	ORL P1, #00001000B	06B5	8900
06B7	99FD	=4893	ANL P1, #NOT 00000010B	ANL P1, #NOT 00000010B	06B7	99FD
06B9	8901	=4894	ORL P1, #00000001B	ORL P1, #00000001B	06B9	8901
06BB	01	=4895	MOVX R1, #R1	MOVX R1, #R1	06BB	01
06BC	2301	=4896	MOV R1, #01H	MOV R1, #01H	06BC	2301
06BE	06C1	=4897	JNI LFEBR1	JNI LFEBR1	06BE	06C1
06C0	27	=4898	CLR R1	CLR R1	06C0	27
06C1	03	=4899	LFEBR1: RET	LFEBR1: RET	06C1	03
		=4900	SIZECHK	SIZECHK		
004A		=4903+	SIZE SET 74	SIZE SET 74	004A	
		=4904+;				
		=4905+;	*****	*****		
		=4914	\$EJECT	\$EJECT		

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUMENT	CHG	LOC	OBJ
		=4915	CODEBLK 85	00000000	0000	0000	0000
0700		=4950+	ORG 1792	00000000	0000	0000	0000
		=4954 ;		00000000	0000	0000	0000
		=4955 ;	LSTORE LOGICAL STORE SUBROUTINE	00000000	0000	0000	0000
		=4956 ;	STORES CONTENTS OF LDATA INTO VARIOUS MEMORY SPACES.	00000000	0000	0000	0000
		=4957 LSTORE:	MMOV A, TYPE	00000000	0000	0000	0000
0700 B937		=4966+	MOV R1, #TYPE	00000000	0000	0000	0000
0702 F1		=4967+	MOV A, @R1	00000000	0000	0000	0000
0703 0306		=4971	ADD A, #LOW LSTTBL	00000000	0000	0000	0000
0705 83		=4972	JMPP @R1	00000000	0000	0000	0000
		=4973 ;		00000000	0000	0000	0000
0706 0C		=4974 LSTTBL:	DB LOW LSTPM	00000000	0000	0000	0000
0707 21		=4975	DB LOW LSTDM	00000000	0000	0000	0000
0708 26		=4976	DB LOW LSTREG	00000000	0000	0000	0000
0709 34		=4977	DB LOW LSTINT	00000000	0000	0000	0000
070A 3D		=4978	DB LOW LSTBRK	00000000	0000	0000	0000
070B 3D		=4979	DB LOW LSTBRK	00000000	0000	0000	0000
		=4980 ;		00000000	0000	0000	0000
		=4981 LSTPM:	MMOV A, SMAHI	00000000	0000	0000	0000
070C B931		=4990+	MOV R1, #SMAHI	00000000	0000	0000	0000
070E F1		=4991+	MOV A, @R1	00000000	0000	0000	0000
070F 9621		=4995	JNZ LSTDM	00000000	0000	0000	0000
		=4996	MMOV A, SMALO	00000000	0000	0000	0000
0711 B930		=5005+	MOV R1, #SMALO	00000000	0000	0000	0000
0713 F1		=5006+	MOV A, @R1	00000000	0000	0000	0000
0714 03E9		=5010	ADD A, #OVSZ	00000000	0000	0000	0000
0716 F621		=5011	JC LSTDM	00000000	0000	0000	0000
		=5012	MMOV A, SMALO	00000000	0000	0000	0000
0718 B930		=5021+	MOV R1, #SMALO	00000000	0000	0000	0000
071A F1		=5022+	MOV A, @R1	00000000	0000	0000	0000
071B 034E		=5026	ADD A, #OVBUF	00000000	0000	0000	0000
071D A9		=5027	MOV R1, A	00000000	0000	0000	0000
071E FA		=5028	MOV A, LDATA	00000000	0000	0000	0000
071F A1		=5029	MOV @R1, A	00000000	0000	0000	0000
0720 83		=5030	RET	00000000	0000	0000	0000
		=5031 ;		00000000	0000	0000	0000
0721 94E1		=5032 LSTDM:	CALL LPGSEL	00000000	0000	0000	0000
0723 FA		=5033	MOV A, LDATA	00000000	0000	0000	0000
0724 91		=5034	MOVX @R1, A	00000000	0000	0000	0000
0725 83		=5035	RET	00000000	0000	0000	0000
		=5036 ;		00000000	0000	0000	0000
		=5037 LSTREG:	MMOV A, SMALO	00000000	0000	0000	0000
0726 B930		=5046+	MOV R1, #SMALO	00000000	0000	0000	0000
0728 F1		=5047+	MOV A, @R1	00000000	0000	0000	0000
0729 537F		=5051	ANL A, #01111111B ; CHECK IF LOW ORDER BITS = 0	00000000	0000	0000	0000
072B C62F		=5052	JZ LSTR0	00000000	0000	0000	0000
072D E4C3		=5053	JMP EPSTOR	00000000	0000	0000	0000
		=5054 ;		00000000	0000	0000	0000
		=5055 LSTR0:	MMOV EPR0, LDATA	00000000	0000	0000	0000
072F FA		=5070+	MOV A, LDATA	00000000	0000	0000	0000
0730 B923		=5084+	MOV R1, #EPR0	00000000	0000	0000	0000
0732 A1		=5085+	MOV @R1, A	00000000	0000	0000	0000
0733 83		=5088	RET	00000000	0000	0000	0000
		=5089 ;		00000000	0000	0000	0000
		=5090 LSTINT:	MMOV A, SMALO	00000000	0000	0000	0000

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	LOC	OBJ
0734	B930	=5099+	MOV R1, #SMALO	11 X10300	0000	0000
0736	F1	=5100+	MOV A, @R1	0000 0000	0000	0000
0737	0320	=5104	ADD A, #EPACC	0000 0000	0000	0000
0739	A9	=5105	MOV R1, A	0000 0000	0000	0000
073A	FA	=5106	MOV A, LDATA	0000 0000	0000	0000
073B	A1	=5107	MOV @R1, A	0000 0000	0000	0000
073C	03	=5108	RET	0000 0000	0000	0000
		=5109 ;				
		=5110 ; LSTBRK LOGICAL STORE OF BREAK-POINT DATA				
073D	94E1	=5111	LSTBRK: CALL LPGSEL	0000 0000	0000	0000
073F	FA	=5112	MOV A, LDATA	0000 0000	0000	0000
0740	1246	=5113	JB0 LSTBR1	0000 0000	0000	0000
0742	8901	=5114	ORL P1, #00000001B	0000 0000	0000	0000
0744	E448	=5115	JMP LSTBR2	0000 0000	0000	0000
0746	99FE	=5116	LSTBR1: ANL P1, #NOT 00000001B	0000 0000	0000	0000
0748	99F7	=5117	LSTBR2: ANL P1, #NOT 00001000B	0000 0000	0000	0000
074A	01	=5118	MOVX A, @R1	0000 0000	0000	0000
074B	0908	=5119	ORL P1, #00001000B	0000 0000	0000	0000
074D	03	=5120	RET	0000 0000	0000	0000
		=5121	SIZECHK			
004E		=5124+ SIZE SET 78				
		=5125+;				
		=5126+; *****				
		=5135 ;				
		=5136 CODEBLK 17				
04E1		=5156+ ORG 1249				
		=5160 ; LPGSEL LOGICAL PAGE SELECT.				
		=5161 ; SETS UP PORT 2 TO ADDRESS APPROPRIATE BYTE OF RAM BLOCK.				
		=5162 LPGSEL: MMOV A, TYPE				
04E1 B937		=5171+ MOV R1, #TYPE				
04E3 F1		=5172+ MOV A, @R1				
04E4 5301		=5176 ANL A, #00000001B ; MASK OFF DATA TYPE SELECTOR BIT				
04E6 47		=5177 SWAP A				
		=5178 MORL A, #SMALI *****				
04E7 B931		=5104+ MOV R1, #SMALI				
04E9 41		=5185+ ORL A, @R1				
04EH 4340		=5189 ORL A, #01000000B				
04EC 3A		=5190 OUTL P2, A				
		=5191 MMOV A, SMALO				
04ED B930		=5200+ MOV R1, #SMALO				
04EF F1		=5201+ MOV A, @R1				
04F0 A9		=5205 MOV R1, A				
04F1 03		=5206 RET				
		=5207 THE SIZECHK				
0011		=5210+ SIZE SET 17				
		=5211+;				
		=5212+; *****				
		=5221 ;				
		=5222 \$EJECT				

LUC OBJ	LINE	SOURCE STATEMENT	STATEMENT	LINE	LOC OBJ
	=5223	CODEBLK 11			
01F2	=5233+	ORG 498			
	=5237	INCSMA INCREMENT STARTING MEMORY ADDRESS WORD.			
01F2 B930	=5238	INCSMA: MOV R1, #SMALO			
01F4 11	=5239	INCH: INC R1			
01F5 F1	=5240	MOV A, R1			
01F6 96FC	=5241	JNZ INCH1			
01F8 19	=5242	INC R1			
01F9 F1	=5243	MOV A, R1			
01FA 17	=5244	INC A			
01FB 31	=5245	XCHD A, R1			
01FC 83	=5246	INCH1: RET			
	=5247	SIZECHK			
0008	=5250+	SIZE SET 11			
	=5251+				
	=5252+	*****			
	=5261				
	=5262	CODEBLK 12			
02F4	=5277+	ORG 756			
	=5281	DECSMA DECREMENT SMA WORD.			
02F4 B930	=5282	DECSMA: MOV R1, #SMALO			
02F6 F1	=5283	MOV A, R1			
02F7 07	=5284	DEC A			
02F8 21	=5285	XCH A, R1			
02F9 96FF	=5286	JNZ DECSM1			
02FB 19	=5287	INC R1			
02FC F1	=5288	MOV A, R1			
02FD 07	=5289	DEC A			
02FE 31	=5290	XCHD A, R1			
02FF 83	=5291	DECSM1: RET			
	=5292	SIZECHK			
000C	=5295+	SIZE SET 12			
	=5296+				
	=5297+	*****			
	=5306				
	=5307	CODEBLK 15			
05E2	=5332+	ORG 1506			
	=5336	CMPMAS COMPARE MEMORY ADDRESSES			
	=5337	COMPARE SMA BYTES WITH EMA BYTES TO DETERMINE RELATIVE MAGNITUDE.			
	=5338	RETURNS WITH CARRY=1 IFF <SMA> >= <EMA>			
	=5339	IS CALLED AFTER ACTION HAS BEEN PERFORMED ON <SMA> TO DETERMINE IF			
	=5340	TASK IS COMPLETED:			
	=5341	IF CY=0 THEN <SMA> >= <EMA> ==> TERMINATE TASK.			
	=5342	IF CY=1 THEN <SMA> < <EMA> ==> INC SMA AND REPEAT.			
	=5343	CMPMAS: MMOV A, SMALO			
05E2 B930	=5352+	MOV R1, #SMALO			
05E4 F1	=5353+	MOV A, R1			
05E5 37	=5357	CPL A			
	=5358	MADD A, EMALO			
05E6 B932	=5364+	MOV R1, #EMALO			
05E8 61	=5365+	ADD A, R1			
	=5369	MMOV A, SMAHI			
05E9 B931	=5370+	MOV R1, #SMAHI			
05EB F1	=5379+	MOV A, R1			
05EC 37	=5383	CPL A			

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTY TWO SOURCE	LINE	LOC	OBJ
		=5384	MADD C A, EMHAI	MOV R0, R1	5384		
05ED	B933	=5390+	MOV R1, #EMHAI	MOV R1, #EMHAI	5390		
05EF	71	=5391+	ADD C A, R1	ADD R0, R1	5391		
05F0	83	=5395	CMPT: RET	RET	5395		
		=5396	SIZECHK	MOV R0, R1	5396		
000F		=5399+	SIZE SET 15	MOV R0, R1	5399		
		=5400+		MOV R0, R1	5400		
		=5401+	*****	MOV R0, R1	5401		
		=5410	\$EJECT	MOV R0, R1	5410		
				MOV R0, R1	5411		
				MOV R0, R1	5412		
				MOV R0, R1	5413		
				MOV R0, R1	5414		
				MOV R0, R1	5415		
				MOV R0, R1	5416		
				MOV R0, R1	5417		
				MOV R0, R1	5418		
				MOV R0, R1	5419		
				MOV R0, R1	5420		
				MOV R0, R1	5421		
				MOV R0, R1	5422		
				MOV R0, R1	5423		
				MOV R0, R1	5424		
				MOV R0, R1	5425		
				MOV R0, R1	5426		
				MOV R0, R1	5427		
				MOV R0, R1	5428		
				MOV R0, R1	5429		
				MOV R0, R1	5430		
				MOV R0, R1	5431		
				MOV R0, R1	5432		
				MOV R0, R1	5433		
				MOV R0, R1	5434		
				MOV R0, R1	5435		
				MOV R0, R1	5436		
				MOV R0, R1	5437		
				MOV R0, R1	5438		
				MOV R0, R1	5439		
				MOV R0, R1	5440		
				MOV R0, R1	5441		
				MOV R0, R1	5442		
				MOV R0, R1	5443		
				MOV R0, R1	5444		
				MOV R0, R1	5445		
				MOV R0, R1	5446		
				MOV R0, R1	5447		
				MOV R0, R1	5448		
				MOV R0, R1	5449		
				MOV R0, R1	5450		
				MOV R0, R1	5451		
				MOV R0, R1	5452		
				MOV R0, R1	5453		
				MOV R0, R1	5454		
				MOV R0, R1	5455		
				MOV R0, R1	5456		
				MOV R0, R1	5457		
				MOV R0, R1	5458		
				MOV R0, R1	5459		
				MOV R0, R1	5460		
				MOV R0, R1	5461		
				MOV R0, R1	5462		
				MOV R0, R1	5463		
				MOV R0, R1	5464		
				MOV R0, R1	5465		
				MOV R0, R1	5466		
				MOV R0, R1	5467		
				MOV R0, R1	5468		
				MOV R0, R1	5469		
				MOV R0, R1	5470		
				MOV R0, R1	5471		
				MOV R0, R1	5472		
				MOV R0, R1	5473		
				MOV R0, R1	5474		
				MOV R0, R1	5475		
				MOV R0, R1	5476		
				MOV R0, R1	5477		
				MOV R0, R1	5478		
				MOV R0, R1	5479		
				MOV R0, R1	5480		
				MOV R0, R1	5481		
				MOV R0, R1	5482		
				MOV R0, R1	5483		
				MOV R0, R1	5484		
				MOV R0, R1	5485		
				MOV R0, R1	5486		
				MOV R0, R1	5487		
				MOV R0, R1	5488		
				MOV R0, R1	5489		
				MOV R0, R1	5490		
				MOV R0, R1	5491		
				MOV R0, R1	5492		
				MOV R0, R1	5493		
				MOV R0, R1	5494		
				MOV R0, R1	5495		
				MOV R0, R1	5496		
				MOV R0, R1	5497		
				MOV R0, R1	5498		
				MOV R0, R1	5499		
				MOV R0, R1	5500		
				MOV R0, R1	5501		
				MOV R0, R1	5502		
				MOV R0, R1	5503		
				MOV R0, R1	5504		
				MOV R0, R1	5505		
				MOV R0, R1	5506		
				MOV R0, R1	5507		
				MOV R0, R1	5508		
				MOV R0, R1	5509		
				MOV R0, R1	5510		
				MOV R0, R1	5511		
				MOV R0, R1	5512		
				MOV R0, R1	5513		
				MOV R0, R1	5514		
				MOV R0, R1	5515		
				MOV R0, R1	5516		
				MOV R0, R1	5517		
				MOV R0, R1	5518		
				MOV R0, R1	5519		
				MOV R0, R1	5520		
				MOV R0, R1	5521		
				MOV R0, R1	5522		
				MOV R0, R1	5523		
				MOV R0, R1	5524		
				MOV R0, R1	5525		
				MOV R0, R1	5526		
				MOV R0, R1	5527		
				MOV R0, R1	5528		
				MOV R0, R1	5529		
				MOV R0, R1	5530		
				MOV R0, R1	5531		
				MOV R0, R1	5532		
				MOV R0, R1	5533		
				MOV R0, R1	5534		
				MOV R0, R1	5535		
				MOV R0, R1	5536		
				MOV R0, R1	5537		
				MOV R0, R1	5538		
				MOV R0, R1	5539		
				MOV R0, R1	5540		
				MOV R0, R1	5541		
				MOV R0, R1	5542		
				MOV R0, R1	5543		
				MOV R0, R1	5544		
				MOV R0, R1	5545		
				MOV R0, R1	5546		
				MOV R0, R1	5547		
				MOV R0, R1	5548		
				MOV R0, R1	5549		
				MOV R0, R1	5550		
				MOV R0, R1	5551		
				MOV R0, R1	5552		
				MOV R0, R1	5553		
				MOV R0, R1	5554		
				MOV R0, R1	5555		
				MOV R0, R1	5556		
				MOV R0, R1	5557		
				MOV R0, R1	5558		
				MOV R0, R1	5559		
				MOV R0, R1	5560		
				MOV R0, R1	5561		
				MOV R0, R1	5562		
				MOV R0, R1	5563		
				MOV R0, R1	5564		
				MOV R0, R1	5565		
				MOV R0, R1	5566		
				MOV R0, R1	5567		
				MOV R0, R1	5568		
				MOV R0, R1	5569		
				MOV R0, R1	5570		
				MOV R0, R1	5571		
				MOV R0, R1	5572		
				MOV R0, R1	5573		
				MOV R0, R1	5574		
				MOV R0, R1	5575		
				MOV R0, R1	5576		
				MOV R0, R1	5577		
				MOV R0, R1	5578		
				MOV R0, R1	5579		
				MOV R0, R1	5580		
				MOV R0, R1	5581		
				MOV R0, R1	5582		
				MOV R0, R1	5583		
				MOV R0, R1	5584		
				MOV R0, R1	5585		
				MOV R0, R1	5586		
				MOV R0, R1	5587		
				MOV R0, R1	5588		
				MOV R0, R1	5589		
				MOV R0, R1	5590		
				MOV R0, R1	5591		
				MOV R0, R1	5592		
				MOV R0, R1	5593		
				MOV R0, R1	5594		
				MOV R0, R1	5595		
				MOV R0, R1	5596		
				MOV R0, R1	5597		
				MOV R0, R1	5598		
				MOV R0, R1	5599		
				MOV R0, R1	5600		
				MOV R0, R1	5601		
				MOV R0, R1	5602		
				MOV R0, R1	5603		
				MOV R0, R1	5604		
				MOV R0, R1	5605		
				MOV R0, R1	5606		
				MOV R0, R1	5607		
				MOV R0, R1	5608		
				MOV R0, R1	5609		
				MOV R0, R1	5610		
				MOV R0, R1	5611		
				MOV R0, R1	5612		
				MOV R0, R1	5613		
				MOV R0, R1	5614		
				MOV R0, R1	5615		
				MOV R0, R1	5616		
				MOV R0, R1	5617		
				MOV R0, R1	5618		
				MOV R0, R1	5619		
				MOV R0, R1	5620		
				MOV R0, R1	5621		
				MOV R0, R1	5622		
				MOV R0, R1	5623		
				MOV R0, R1	5624		
				MOV R0, R1	5625		
				MOV R0, R1	5626		
				MOV R0, R1	5627		
				MOV R0, R1	5628		
				MOV R0, R1	5629		
				MOV R0, R1	5630		
				MOV R0, R1	5631		
				MOV R0, R1	5632		
				MOV R0, R1	5633		
				MOV R0, R1	5634		
				MOV R0, R1	5635		
				MOV R0, R1	5636		
				MOV R0, R1	5637		
				MOV R0, R1	5638		
				MOV R0, R1	5639		
				MOV R0, R1	5640		
				MOV R0, R1	5641		
				MOV R0, R1	5642		
				MOV R0, R1	56		

LOC	OBJ	LINE	SOURCE STATEMENT	THESISTZ	CONV	3M1J	130	30J
		5411 \$	INCLUDE(:F0:KBD.MOD)	THMCL A	3000H	N002=		
		=5412	CODEBLK 100	THMCL JS	Y01	+002=	000	000
074E		=5447+	ORG 1070	THMCL A	3000	+002=	17	000
		=5451 ;		THMCL JS	Y01	+002=	00	000
		=5452 ;	KEYBOARD AND DISPLAY PROCESSING ROUTINE	THMCL JS	Y01	+002=		
		=5453 ;	CALLED PERIODICALLY WHEN KBD AND DISPLAY ARE TO BE ALIVE	THMCL JS	Y01	+002=		
074E D5		=5454 TIINT:	SEL RB1	THMCL JS	Y01	+002=		
		=5455	MNOV ASAVE, A	THMCL JS	Y01	+002=		
074F B93E		=5468+	MOV R1, #ASAVE	THMCL JS	Y01	+002=		
0751 R1		=5469+	MOV @R1, A	THMCL JS	Y01	+002=		
0752 23F0		=5473	MOV A, #(-10H)	THMCL JS	Y01	+002=		
0754 62		=5474	MOV T, A ;RELOAD TIMER INTERVAL	THMCL JS	Y01	+002=		
0755 27		=5475	CLR A	THMCL JS	Y01	+002=		
0756 3E		=5476	MOVD PSEGH1, A ;WRITE BLANK PATTERN TO SEG DRIVERS	THMCL JS	Y01	+002=		
0757 3D		=5477	MOVD PSEGLO, A	THMCL JS	Y01	+002=		
0758 FD		=5478	MOV A, CURDIG	THMCL JS	Y01	+002=		
0759 07		=5479	DEC A	THMCL JS	Y01	+002=		
075A 3F		=5480	MOVD PDIGIT, A ;ENERGIZE CHARACTER	THMCL JS	Y01	+002=		
075B 0C		=5481	MOVD A, PINPUT ;LOAD ANY SWITCH CLOSURES	THMCL JS	Y01	+002=		
075C AA		=5482	MOV ROTPAT, A	THMCL JS	Y01	+002=		
		=5483	;WRITE NEXT SEGMENT PATTERN	THMCL JS	Y01	+002=		
075D FD		=5484	MOV A, CURDIG	THMCL JS	Y01	+002=		
075E 07		=5485	DEC A	THMCL JS	Y01	+002=		
075F 0346		=5486	ADD A, #SEGMAP ;ADD CURDIG DISPLACEMENT TO BASE	THMCL JS	Y01	+002=		
0761 A8		=5487	MOV R0, A	THMCL JS	Y01	+002=		
0762 F0		=5488	MOV A, @R0 ;LOAD ACC W/ NEXT SEGMENT PATTERN	THMCL JS	Y01	+002=		
0763 3D		=5489	MOVD PSEGLO, A ;ENABLE APPROPRIATE SEGMENTS	THMCL JS	Y01	+002=		
0764 47		=5490	SWAP A	THMCL JS	Y01	+002=		
0765 3E		=5491	MOVD PSEGH1, A	THMCL JS	Y01	+002=		
		=5492 ;		THMCL JS	Y01	+002=		
		=5493 ;	*****	THMCL JS	Y01	+002=		
		=5494 ;	THE NEXT CHARACTER IS NOW BEING DISPLAYED.	THMCL JS	Y01	+002=		
		=5495 ;	THE KEYBOARD SCAN ROUTINE IS INTEGRATED INTO THE DISPLAY SCAN.	THMCL JS	Y01	+002=		
		=5496 ;	WITH THE CURRENT ROW ENERGIZED, CHECK IF THERE ARE ANY INPUTS.	THMCL JS	Y01	+002=		
		=5497 ;	*****	THMCL JS	Y01	+002=		
		=5498 ;		THMCL JS	Y01	+002=		
		=5499 ;	ROTATE BITS THROUGH THE CY WHILE INCREMENTING KEYLOC.	THMCL JS	Y01	+002=		
		=5500 ;		THMCL JS	Y01	+002=		
0766 B804		=5501	MOV ROTCNT, #NCOLS ;SET UP FOR <NCOLS> LOOPS THROUGH 'NXTLOC'	THMCL JS	Y01	+002=		
		=5502 NXTLOC:	MRRC ROTPAT	THMCL JS	Y01	+002=		
0768 FA		=5514+	MOV A, ROTPAT	THMCL JS	Y01	+002=		
0769 67		=5518+	RRC A	THMCL JS	Y01	+002=		
076A AA		=5529+	MOV ROTPAT, A	THMCL JS	Y01	+002=		
076B F68B		=5532	JC SCANS ;ONE BIT IN CY INDICATES KEY NOT DOWN	THMCL JS	Y01	+002=		
076D BE01		=5533	MOV KEYFLG, #1 ;MARK THAT AT LEAST ONE KEY WAS DETECTED	THMCL JS	Y01	+002=		
		=5534	; \ IN THE CURRENT SCAN	THMCL JS	Y01	+002=		
		=5535 ;		THMCL JS	Y01	+002=		
		=5536 ;	*****	THMCL JS	Y01	+002=		
		=5537 ;	A KEYSTROKE WAS DETECTED FOR THE CURRENT COLUMN. ITS	THMCL JS	Y01	+002=		
		=5538 ;	POSITION IS IN REGISTER KEYLOC. SEE IF SAME KEY SENSED LAST CYCLE.	THMCL JS	Y01	+002=		
		=5539 ;	*****	THMCL JS	Y01	+002=		
		=5540 ;		THMCL JS	Y01	+002=		
		=5541	MNOV A, KEYLOC	THMCL JS	Y01	+002=		
076F B93C		=5550+	MOV R1, #KEYLOC	THMCL JS	Y01	+002=		
0771 F1		=5551+	MOV A, @R1	THMCL JS	Y01	+002=		

LOC	OBJ	LINE	SOURCE STATEMENT	THAT STATE	LINE	LOC
0772	2C	=5555	XCH A, LASTKY			
0773	DC	=5556	XRL A, LASTKY			
0774	C67C	=5557	JZ SCAN3			
		=5558 ;				
		=5559 ;	*****			
		=5560 ;	A DIFFERENT KEY WAS READ ON THIS CYCLE THAN ON THE PREVIOUS CYCLE			
		=5561 ;	SET NREPTS TO THE DEBOUNCE PARAMETER FOR A NEW COUNTDOWN			
		=5562 ;	*****			
		=5563 ;				
0776	B93D	=5564	MOV R1, #NREPTS			
0778	B106	=5565	MOV @R1, #6			
077A	E48B	=5566	JMP SCAN5			
		=5567 ;				
		=5568 ;	*****			
		=5569 ;	SAME KEY WAS DETECTED 35 ON PREVIOUS CYCLE			
		=5570 ;	LOOK AT NREPTS: IF ALREADY ZERO, DO NOTHING			
		=5571 ;	ELSE DECREMENT NREPTS			
		=5572 ;	IF THIS RESULTS IN ZERO, MOVE LASTKY INTO KDBBUF			
		=5573 ;	*****			
		=5574 ;				
		=5575 SCAN3:	MMOV A, NREPTS			
077C	B93D	=5584+	MOV R1, #NREPTS			
077E	F1	=5585+	MOV A, @R1			
077F	C68B	=5589	JZ SCAN5 ; IF ALREADY ZERO			
0781	07	=5590	DEC A ; INDICATE ONE MORE SUCCESSIVE KEY DETECTION			
		=5591	MMOV NREPTS, A			
0782	B93D	=5604+	MOV R1, #NREPTS			
0784	A1	=5605+	MOV @R1, A			
0785	968B	=5609	JNZ SCAN5 ; IF DECREMENT DOES NOT RESULT IN ZERO			
		=5610	MMOV KDBBUF, LASTKY ; TO MARK NEW KEY CLOSURE			
0787	FC	=5633+	MOV A, LASTKY			
0788	B93B	=5639+	MOV R1, #KDBBUF			
078A	A1	=5640+	MOV @R1, A			
		=5643 ;				
078B	B93C	=5644	SCAN5: MOV R1, #KEYLOC			
078D	11	=5645	INC @R1			
078E	ED68	=5646	DJNZ ROTCNT, NXTLOC			
0790	EDAS	=5647	DJNZ CURDIG, TIRET1			
0792	B00B	=5648	MOV CURDIG, #CHARNO			
		=5649 ;				
		=5650 ;	*****			
		=5651 ;	THE FOLLOWING CODE SEGMENT IS USED BY THE KEYBOARD SCANNING ROUTINE			
		=5652 ;	IT IS EXECUTED ONLY AFTER A REFRESH SEQUENCE IS COMPLETED			
		=5653 ;	*****			
		=5654 ;				
		=5655	MMOV KEYLOC, ZERO			
0794	B93C	=5666+	MOV R1, #KEYLOC			
0796	B100	=5667+	MOV @R1, #ZERO			
0798	FE	=5671	MOV A, KEYFLG			
0799	969D	=5672	JNZ SCAN8 ; JUMP IF ANY KEYS WERE DETECTED			
		=5673	MMOV LASTKY, NEG1 ; CHANGE <LASTKY> WHEN NO KEYS ARE DOWN			
079B	BCFF	=5678+	MOV LASTKY, #NEG1			
079D	BE00	=5682	SCAN8: MOV KEYFLG, #0			
		=5683 ;				
		=5684 ;	*****			

LOC	OBJ	LINE	SOURCE STATEMENT	INSTR	PC	LOC
		=5685 ;		MOV	0000	0000
		=5686 ;	KBD/DISP RETURN CODE- RESTORES SYSTEM STATUS.	MOV	0000	0000
		=5687	MMOV A, RDELAY	MOV	0000	0000
079F	B93F	=5696+	MOV R1, #RDELAY	MOV	0000	0000
07A1	F1	=5697+	MOV A, @R1	MOV	0000	0000
07A2	06A8	=5701	JZ TIRET1	JZ	0000	0000
07A4	07	=5702	DEC A	DEC	0000	0000
		=5703	MMOV KDELAY, A	MOV	0000	0000
07A5	B93F	=5716+	MOV R1, #RDELAY	MOV	0000	0000
07A7	A1	=5717+	MOV @R1, A	MOV	0000	0000
		=5721 TIRET1:	MMOV A, @SAVE	MOV	0000	0000
07A8	B93E	=5730+	MOV R1, #ASAVE	MOV	0000	0000
07AA	F1	=5731+	MOV A, @R1	MOV	0000	0000
07AB	93	=5735	RETR	RETR	0000	0000
		=5736 ;			0000	0000
		=5737 ;			0000	0000
		=5738 ;	TOFFOL TIMER OVERFLOW POLLING SUBROUTINE.		0000	0000
		=5739 ;	CALLLED REPEATEDLY FROM WHEREVER KBD/DISP MUST BE ALIVE.		0000	0000
		=5740 ;	MONITORS THE TIMER OVERFLOW FLAG (TOF) AND CALLS SERVICE		0000	0000
		=5741 ;	ROUTINE WHEN APPROPRIATE.		0000	0000
07AC	164E	=5742 TUFFOL:	JTF TIINT.	JTF	0000	0000
07AE	83	=5743	RET	RET	0000	0000
		=5744	SIZECHK		0000	0000
0061		=5747+ SIZE SET 97	SET 97	SET	0000	0000
		=5748+;			0000	0000
		=5749+;*****			0000	0000
		=5758 \$EJECT			0000	0000

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	LINE	LOC	OBJ
		=5759	CODEBLK 17	00000000	0000		
06C2		=5759+	ORG 1730	00000000	0000		
		=5793 ;					
		=5794 ; KBDIN	KEYBOARD INPUT SUBROUTINE.				
		=5795 ;	RETURNS ONLY AFTER A NEW KEYSTROKE HAS BEEN DETECTED AND DEBOUNCED.				
		=5796 ;	VALUE OF KEY POSITION IN SWITCH MATRIX IS				
		=5797 ;	RETURNED IN THE ACCUMULATOR.				
		=5798 ;	DISPLAY CHARACTER NOW ON BLANKED BEFORE RETURNING.				
06C2 DF03		=5799 KBDIN:	MOV XPCODE, #3				
06C4 74D1		=5800	CALL XPTST				
06C6 F4AC		=5801 KBD11:	CALL TOPPOL				
		=5802	MMOV A, KBDDEF				
06C8 B938		=5811:	MOV R1, #KBDDEF				
06CA F1		=5812:	MOV A, R1				
06CB F2C6		=5816	JB7 KBD11				
06CD 27		=5817	CLR A				
06CE 3E		=5818	MOVD PSEGL1, A				
06CF 3D		=5819	MOVD PSEGL0, A				
06D0 37		=5820	CPLM A				
06D1 21		=5821	XCH A, R1				
06D2 83		=5822	RET				
		=5823	SIZECHK				
0011		=5826+ SIZE SET 17					
		=5827+;					
		=5828+; *****					
		=5837 ;					
		=5838	CODEBLK 15				
05F1		=5863+	ORG 1512				
		=5867 ; CLEAR	WRITES 'BLANK' CHARACTERS INTO ALL DISPLAY REGISTERS.				
		=5868 ;	RETURNS WITH NEXTPL SET TO LEFTMOST CHARACTER POSITION				
		=5869 ;	DOES NOT AFFECT ACC OR CYCLES				
05F1 B846		=5870 CLEAR:	MOV T10R0, #SEGMAP1				
05F3 B908		=5871	MOV T10R1, #CHARNO1				
05F5 B000		=5872 DBLANK:	MOV T10R0, #0				
05F7 18		=5873	INC T10R0				
05F8 E9F5		=5874	DJNZ T10R1, DBLANK1				
		=5875	MMOVT10NEXTPL, CHARNO				
05FA D93A		=5886+	MOV R1, #NEXTPL				
05FC D108		=5897+	MOV R1, R1, #CHARNO				
05FE 83		=5891	RET				
		=5892	SIZECHK				
000E		=5895+ SIZE SET 14					
		=5896+;					
		=5897+; *****					
		=5906 ;					
		=5907	CODECLK 44				
06D3		=5937+	ORG 1747				
		=5941 ; DSPACC	DISPLAY VALUE OF LOW NIBBLE OF ACC				
06D3 530F		=5942 DSPACC:	ANL A, #0FH				
06D5 03EF		=5943	ADD A, #DGPRTS				
06D7 A3		=5944	MOVPRM A, R1				
		=5945 ; DISP	WRITES BIT PATTERN NOW IN ACC INTO NEXT CHARACTER POSITION				
		=5946 ;	OF THE DISPLAY (NEXTPL). INCREMENTS NEXTPL				
		=5947 ;	RESULTS IN DISPLAY BEING FILLED LEFT TO RIGHT, THEN RESTARTING				
06D8 AE		=5948 DISP:	MOV DSPTMP, A				

```

00D9 BF04 =5949 MOV XPCODE, #4
00DB 74D1 =5950 CALL XPTST
          =5951 MMOV R, NEXTPL
00DD B93A =5960+ MOV R1, #NEXTPL
00DF F1 =5961+ MOV R, R1
00E0 0345 =5965 ADD R, #SEGMAP-1
00E2 A9 =5966 MOV R1, A
00E3 FE =5967 MOV R, DSPTMP
00E4 A1 =5968 MOV R1, A
          =5969 MDJNZ NEXTPL, WDISP1
00E5 B93A =5974+ MOV R1, #NEXTPL
00E7 F1 =5975+ MOV R, R1
00E8 07 =5979+ DEC R
00E9 A1 =5984+ MOV R1, A
00EA 96EE =5988+ JNZ WDISP1
00EC D108 =5990 MOV R1, #CHARNO
00EE 83 =5991 WDISP1: RET
          =5992 ;
          =5993 ; DGPATS IS THE BASE FOR THE TABLE OF SEGMENT PATTERNS FOR HEX DIGITS.
          =5994 ; HERE THE FULL HEX SET (0-F) IS INCLUDED.
          =5995 ;
00EF =5996 DGPATS EQU $ AND OFFH
          =5997 ;
          =5998 ; FORMAT IS PGFDCBA IN STANDARD SEVEN-SEGMENT ENCODING CONVENTION
          =5999 ; WHERE P REPRESENTS THE DECIMAL POINT
00EF 3F =6000 DB 00111111B ; SEGMENT PATTERN FOR DIGIT '0'
00F0 06 =6001 DB 00000110B ; SEGMENT PATTERN FOR DIGIT '1'
00F1 58 =6002 DB 01011011B ; SEGMENT PATTERN FOR DIGIT '2'
00F2 4F =6003 DB 01001111B ; SEGMENT PATTERN FOR DIGIT '3'
00F3 66 =6004 DB 01100110B ; SEGMENT PATTERN FOR DIGIT '4'
00F4 6D =6005 DB 01101101B ; SEGMENT PATTERN FOR DIGIT '5'
00F5 7D =6006 DB 01111101B ; SEGMENT PATTERN FOR DIGIT '6'
00F6 07 =6007 DB 00000111B ; SEGMENT PATTERN FOR DIGIT '7'
00F7 7F =6008 DB 01111111B ; SEGMENT PATTERN FOR DIGIT '8'
00F8 67 =6009 DB 01100111B ; SEGMENT PATTERN FOR DIGIT '9'
00F9 77 =6010 DB 01110111B ; SEGMENT PATTERN FOR DIGIT 'A'
00FA 7C =6011 DB 01111100B ; SEGMENT PATTERN FOR DIGIT 'B'
00FB 39 =6012 DB 00111001B ; SEGMENT PATTERN FOR DIGIT 'C'
00FC 5E =6013 DB 01011100B ; SEGMENT PATTERN FOR DIGIT 'D'
00FD 79 =6014 DB 01111001B ; SEGMENT PATTERN FOR DIGIT 'E'
00FE 71 =6015 DB 01110001B ; SEGMENT PATTERN FOR DIGIT 'F'
          =6016 SIZECHK
002C =6019+ SIZE SET 44
          =6020+ ;
          =6021+ ; *****
          =6030 ;
          =6031 CODEBLK 12
04F2 =6051+ ORG 1266
          =6055 ; DELAY SUBROUTINE WAITS FOR THE NUMBER OF COMPLETE
          =6056 ; DISPLAY SCANS CORRESPONDING TO THE ACC CONTENTS.
          =6057 ; USED WITH CRUDE HUMAN INTERFACES- AS WHEN OPERATOR SHOULD SEE
          =6058 ; SOME DISPLAY CHANGE WHILE IT IS CHANGING.
          =6059 DELAY: MMOV R, DELAY, A
          =6072+ MOV R1, #DELAY
04F2 B93F =6073+ MOV R1, A
04F4 A1

```


LOC	OBJ	LINE	SOURCE STATEMENT	THIRDS	THIRDS	LOC	OBJ
04F5	F4AC	=6077	DELAY1: CALL 10FFPOL				
		=6078	MMOV A, RDELAY				
04F7	093F	=6087+	MOV R1, #RDELAY				
04F9	F1	=6088+	MOV A, R1				
04FA	96F5	=6092	JNZ DELAY1				
04FC	83	=6093	RET				
		=6094	SIZECHK				
0008		=6097+	SIZE SET 11				
		=6098+					
		=6099+	*****				
		=6100					
		=6109	CODEBLK 8				
07FF		=6144+	ORG 1967				
		=6148	;KBDPOL POLL STATUS OF KEYBOARD INPUT ROUTINE				
		=6149	; RETURN WITH ACC BIT 7 = 0 IF KEYBOARD INPUT HAS BEEN RECEIVED.				
07FF	BF05	=6150	KBDPOL: MOV XPCODE, #5				
07B1	74D1	=6151	CALL XPTST				
		=6152	MMOV A, KBDDBUF				
07B3	B93B	=6161+	MOV R1, #KBDDBUF				
07D5	F1	=6162+	MOV A, R1				
07B6	83	=6166	RET				
		=6167	SIZECHK				
0008		=6170+	SIZE SET 8				
		=6171+					
		=6172+	*****				
		=6181	\$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	STATEMENT	LINE	LOC
		6182 \$	INCLUDE(.F0:LINK.MOD)	107701	1180	100
		=6183	CODEBLK 15	107701	1180	100
07B7		=6218+	ORG 1975	107701	1180	100
		=6222 ;EPFET	FETCH DATA BYTE FROM EP INTERNAL RAM ADDRESSED BY \$M\$ALO.	107701	1180	100
		=6223 EPFET:	MMOV A, \$M\$ALO	107701	1180	100
07B7 B930		=6232+	MOV R1, \$M\$ALO	107701	1180	100
07B9 F1		=6233+	MOV A, R1	107701	1180	100
07BA F4D0		=6237	CALL EPPASS	107701	1180	100
07BC 2380		=6238	MOV A, \$10000000B	107701	1180	100
07BE F4D0		=6239	CALL EPPASS	107701	1180	100
07C0 F4D0		=6240	CALL EPPASS	107701	1180	100
07C2 83		=6241	RET	107701	1180	100
		=6242	SIZECHK	107701	1180	100
000C		=6245+ SIZE SET 12		107701	1180	100
		=6246+;		107701	1180	100
		=6247+; *****		107701	1180	100
		=6256 ;		107701	1180	100
		=6257	CODEBLK 15	107701	1180	100
07C3		=6292+	ORG 1987	107701	1180	100
		=6296 ;EPSTOR	STORE DATA IN LDATA IN EP INTERNAL RAM AT (\$M\$ALO)	107701	1180	100
07C3 FA		=6297 EPSTOR:	MOV A, LDATA	107701	1180	100
07C4 F4D0		=6298	CALL EPPASS	107701	1180	100
		=6299	MMOV A, \$M\$ALO	107701	1180	100
07C6 B930		=6308+	MOV R1, \$M\$ALO	107701	1180	100
07C8 F1		=6309+	MOV A, R1	107701	1180	100
07C9 537F		=6313	ANL A, \$01111111B	107701	1180	100
07CB F4D0		=6314	CALL EPPASS	107701	1180	100
07CD F4D0		=6315	CALL EPPASS	107701	1180	100
07CF 83		=6316	RET	107701	1180	100
		=6317	SIZECHK	107701	1180	100
000D		=6320+ SIZE SET 13		107701	1180	100
		=6321+;		107701	1180	100
		=6322+; *****		107701	1180	100
		=6331 \$EJECT		107701	1180	100

LOC	OBJ	LINE	SOURCE STATEMENT	INSTR	STAT	COND	VAL
		=6332 ;	THE FOLLOWING UTILITIES INVOLVE INTERCHANGES BETWEEN THE MP AND EP.				
		=6333 ;					
		=6334	CODEBLK 11				
07D0		=6369+	ORG 2000				
		=6373 ;	EPPASS PASSES A SINGLE PARAMETER BYTE TO THE EP THROUGH THE LINK.				
		=6374 ;	WRITE THE CONTENTS OF THE ACC TO THE LINK;				
		=6375 ;	RELEASE THE EP;				
		=6376 ;	READ THE LINK INTO THE ACC;				
		=6377 ;	RETURN.				
07D0 8A30		=6378 EPPASS:	ORL P2, #00110000B ;	ENABLE LINK WRITES.			
07D2 91		=6379	MOVX A, @R1 ;	WRITE ACC TO LINK.			
07D3 99FE		=6380	ANL P1, #NOT ENBRAM ;	DISABLE BREAKPOINTS.			
07D5 0902		=6381	ORL P1, #ENBLNK ;	SET TO BREAK ON LINK REFERENCE.			
07D7 F4DB		=6382	CALL EPSTEP				
07D9 91		=6383	MOVX A, @R1				
07DA 83		=6384	RET				
		=6385	SIZECHK				
0006		=6388+ SIZE SET 11					
		=6389+;					
		=6390+; *****					
		=6399 ;					
		=6400	CODEBLK 25				
07D6		=6435+	ORG 2011				
		=6439 ;	EPSTEP RELEASES EP TO RUN IN PRESENT MODE UNTIL AN ANTICIPATED				
		=6440 ;	HARDWARE BREAK OCCURS.				
		=6441 ;	(DUE TO SINGLE STEPPING, LINK OP-CODE FETCH, OR LINK DATA FETCH.)				
		=6442 ;	MUST OCCUR WITHIN A FINITE NUMBER OF CYCLES ((40 MP CYCLES)				
		=6443 ;	OR WATCHDOG TIMER WILL ASSUME A COMMUNICATIONS ERROR				
		=6444 ;	WATCHDOG BETWEEN THE MP AND EP.				
07D8 F4F4		=6445 EPSTEP:	CALL EPREL				
07D0 B90A		=6446	MOV R1, #10				
07DF 86F1		=6447 EPSTE1:	JNI EPSTE2				
07E1 E9D1		=6448	DJNZ R1, EPSTE1				
07E3 8910		=6449	ORL P1, #EPRSET				
07E5 744F		=6450	CALL EPBRK				
07E7 B88B		=6451	MOV R0, #LOW(OVLBAS+OVSZ)				
07E9 746A		=6452	CALL OVLORD				
07EB 99EF		=6453	ANL P1, #NOT EPRSET				
07ED B08E		=6454	MOV LDAT, #0EH				
07EF 249A		=6455	JMP PERROR				
07F1 744F		=6456 EPSTE2:	CALL EPBRK				
07F3 83		=6457	RET				
		=6458	SIZECHK				
0019		=6461+ SIZE SET 25					
		=6462+;					
		=6463+; *****					
		=6472 ;					
		=6473 ;					
		=6474 \$EJECT					

```

=6475 CODEBLK 9
07F4 =6510+ ORG 2036
=6514 ; EPREL RELEASES EP TO RUN IN PRESENT MODE.
=6515 ; SEQUENCE IS AS FOLLOWS:
=6516 ; PUT MEMORY ARRAY IN EP MODE;
=6517 ; RAISE /SSSTEP;
=6518 ; RETURN.
07F4 99F7 =6519 EPREL: ANL P1, #NOT CLRBF ; CLEAR BREAK F/F.
07F6 8908 =6520 ORL P1, #CLRBF ; RE-ENABLE BREAK F/F.
07F8 9ABF =6521 ANL P2, #NOT 01000000B ; ENABLE EP CONTROL OF MEM ARRAY.
07FA 8904 =6522 ORL P1, #00000100B ; FREE EP TO RUN UNTIL BREAK.
07FC 83 =6523 RET
=6524 SIZECHK
0009 =6527+ SIZE SET 9
=6528+
=6529+ ;*****
=6530 ;
=6531 ;
=6540 CODEBLK 11
034F =6580+ ORG 847
=6584 ; EPBRK REGAIN CONTROL OF MEMORY ARRAY FROM EP.
=6585 ; DROP /SSSTEP;
=6586 ; WAIT 30 USECS. ;
=6587 ; PUT MEMORY ARRAY IN MP MODE;
=6588 ; RETURN.
034F 99FB =6589 EPBRK: ANL P1, #NOT 00000100B ; FREEZE EMULATION PROCESSOR.
0351 8920 =6590 ORL P1, #MODOUT ; SIGNAL EP IS NOT RUNNING USER CODE.
0353 8905 =6591 MOV R1, #5
0355 E955 =6592 DJNZ R1, $ ; DELAY FOR EP TO FINISH INSTRUCTION.
0357 8A40 =6593 ORL P2, #01000000B ; SEIZE CONTROL OF MEM ARRAY.
0359 83 =6594 RET
=6595 SIZECHK
0008 =6598+ SIZE SET 11
=6599+
=6600+ ;*****
=6601 ;
=6610 ;
=6611 CODEBLK 16
035A =6651+ ORG 858
=6655 ; OVSMP OVERLAY SWAP.
=6656 ; SWAPS BLOCK OF DATABYTES (USER'S PROGRAM) BETWEEN MP RAM & EP PM.
035A 8865 =6657 OVSMP: MOV R0, #OVBUF+OVSZ
035C 8917 =6658 MOV R1, #OVSZ
035E 2340 =6659 MOV A, #01000000B
0360 3A =6660 OUTL P2, A
0361 C8 =6661 OVSMP1: DEC R0
0362 C9 =6662 DEC R1
0363 81 =6663 MOVX A, @R1
0364 20 =6664 XCH A, @R0
0365 91 =6665 MOVX @R1, A
0366 F9 =6666 MOV A, R1
0367 9661 =6667 JNZ OVSMP1
0369 83 =6668 RET
=6669 SIZECHK
0010 =6672+ SIZE SET 16

```



```

=6673+;
=6674+; *****
=6683 ;
=6684 CODEBLK 14
036A =6724+ ORG 874
=6728 ; OVLORD OVERLAY LOAD.
=6729 ; MOVES BLOCK OF DATA BYTES (ASSEMBLED SOURCE) FROM PG3 TO EP PM.
=6730 ; TOP OF DATA BLOCK LOADED AND BLOCK LENGTH DETERMINED BY R0 AND R1.
036A B917 =6731 OVLORD: MOV R1, #OVSZIE
036C 2340 =6732 MOV R, #01000000
036E 3A =6733 OUTL F2, R
036F C8 =6734 MML01: DEC R0
0370 C9 =6735 DEC R1
0371 F8 =6736 MOV R, R0
0372 E3 =6737 MOV P3, A, R0
0373 91 =6738 MOVX R0, R1
0374 F9 =6739 MOV R1, R1
0375 966F =6740 JNZ MML01
0377 83 =6741 RET
=6742 SIZECHK
000E =6745+ SIZE SET 14
=6746+;
=6747+; *****
=6756 $EJECT

```

1

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTATZ	DOAGZ	LINE	LOC
		=6757 ;					
		=6758 ;					
		=6759 ;					
		=6760 ;	THE REST OF THIS MODULE CONTAINS THE MINI-MONITORS WHICH OVERLAY				
		=6761 ;	THE EMULATION PROCESSOR PROGRAM RAM TO GIVE THE				
		=6762 ;	MASTER PROCESSOR ACCESS TO INTERNAL REGISTERS AND RAM OF THE EP.				
		=6763 ;					
		=6764 ;					
		=6765 ;					
		=6766	DATABLK 22				
0378		=6771+	ORG 008				
		=6775 ;					
		=6776 ;	OV0- OVERLAY TO BREAK EP EXECUTION AND JUMP TO LOCATION 009H.				
		=6777 ;	LOCATION 009H REACHED WITH TOP-OF-STACK = RETURN ADDRESS+2				
		=6778 ;	DUE TO FORCED "CALL" DURING WHICH PC WAS INCREMENTED.				
		=6779 ;	LOCS 003H & 007H CALL 009H TO SIMULATE SAME CONDITION				
		=6780 ;	IF BREAK OCCURS DURING INTERRUPT CYCLE.				
		=6781 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM				
		=6782 ;					
0378		=6783	OV0BAS EQU \$				
0378		=6784	ORG OV0BAS				
0378 1409		=6785	CALL 009H				
037A 00		=6786	NOP				
		=6787 ;					
037B		=6788	ORG OV0BAS+003H				
037B 1409		=6789	CALL 009H				
037D 00		=6790	NOP				
037E 00		=6791	NOP				
		=6792 ;					
037F		=6793	ORG OV0BAS+007H				
037F 1409		=6794	CALL 009H				
0381 00		=6795	NOP				
0382 00		=6796	NOP				
0383 00		=6797	NOP				
0384 00		=6798	NOP				
0385 00		=6799	NOP				
0386 00		=6800	NOP				
0387 00		=6801	NOP				
0388 00		=6802	NOP				
0389 00		=6803	NOP				
038A 00		=6804	NOP				
038B 00		=6805	NOP				
		=6806 ;					
038C		=6807	ORG OV0BAS+014H				
038C 0409		=6808	JMP 009H				
		=6809 ;					
		=6810	SIZECHK				
0016		=6813+	SIZE SET 22				
		=6814+;					
		=6815+;	*****				
		=6824	\$EJECT				

	=6883	DATABLK 22			
03A4	=6884	ORG 932			
	=6892 ;				
	=6893 ;	OV1-	OVERLAY 1 TO GIVE MP ACCESS TO EP RAM LOCS. 01H-7FH		
	=6894 ;		SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM		
	=6895 ;				
03A4	=6896	OV1BAS EQU	\$		
	=6897 ;				
03A4 040A	=6898	JMP	OV1B1		
03A6 00	=6899	NOP			
	=6900 ;				
03A7	=6901	ORG	OV1BAS+003H		
03A7 83	=6902	RET			
03A8 00	=6903	NOP			
03A9 00	=6904	NOP			
03AN 00	=6905	NOP			
	=6906 ;				
03AB	=6907	ORG	OV1BAS+007H		
03AB 83	=6908	RET			
03AC 00	=6909	NOP			
	=6910 ;				
03AD	=6911	ORG	OV1BAS+009H		
03AD 90	=6912	MOVX	@R0, A		
	=6913 ;				
000A	=6914	OV1B1 EQU	\$-OV1BAS		
	=6915 ;				
03AE 00	=6916	MOVX	A, @R0		
03AF A8	=6917	MOV	R0, A		
03B0 00	=6918	MOVX	A, @R0		
03B1 F213	=6919	JB7	OV1B2		
03B3 20	=6920	XCH	A, R0		
03B4 A0	=6921	MOV	@R0, A		
03B5 0409	=6922	JMP	005H		
	=6923 ;				
0313	=6924	OV1B2 EQU	\$-LOW OV1BAS		
	=6925 ;				
03B7 F0	=6926	MOV	A, @R0		
03B8 0409	=6927	JMP	005H		
	=6928 ;				
	=6929	SIZECHK			
0016	=6932	SIZE SET	22		
	=6933+;				
	=6934+;	*****			
	=6943	\$EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	THINKSTATE	LINE	LOC	OBJ
		=6944	DATABLK 23				
03DA		=6949+	ORG 954				
		=6953 ;					
		=6954 ; OV2-	OVERLAY TO RESTORE EP STATUS SAVED ON BREAK AND RESUME USER'S PROGRAM.				
		=6955 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.				
		=6956 ;					
03BA		=6957 OV2BAS	EQU \$				
03BH		=6958 ORG	OV2BAS				
03BA 0400		=6959	JMP 000H				
03BC 00		=6960	NOP				
		=6961 ;					
03BD		=6962 ORG	OV2BAS+003H				
03BD 03		=6963	RET				
03BE 00		=6964	NOP				
03BF 00		=6965	NOP				
03C0 00		=6966	NOP				
		=6967 ;					
03C1		=6968 ORG	OV2BAS+007H				
03C1 03		=6969	RET				
03C2 00		=6970	NOP				
		=6971 ;					
03C3		=6972 ORG	OV2BAS+009H				
03C3 90		=6973	MOVX 0R0, A				
		=6974 ;					
03C4 00		=6975	MOVX A, 0R0				
03C5 A8		=6976	MOV R0, A				
03C6 00		=6977	MOVX A, 0R0				
03C7 D7		=6978	MOV I\$M, A				
03C8 A5		=6979	CLR F1				
03C9 B5		=6980	CPL F1				
03CA 7213		=6981	JB3 OV2B1				
03CC A5		=6982	CLR F1				
		=6983 ;					
0313		=6984 OV2B1	EQU \$-LOW OV2BAS				
		=6985 ;					
03CD 00		=6986	MOVX A, 0R0				
03CE 62		=6987	MOV T, A				
03CF 00		=6988	MOVX A, 0R0				
03D0 93		=6989	RETR				
		=6990	SIZECHK				
0017		=6993+ SIZE SET 23					
		=6994+;					
		=6995+; *****					
		=7004 \$EJECT					

1

LOC	OBJ	LINE	SOURCE STATEMENT	INDICATE	GROUP	VAL	LOC	OBJ
		7005 ;						
		7006	CODEBLK 11					
03D1		7046+	ORG 977					
03D1 0A00		7050	XPTEST: ORL P2, #00H					
03D3 0A		7051	IN R, P2					
03D4 9A7F		7052	ANL P2, #(NOT 00H)					
03D6 F2D9		7053	JB7 \$+3					
03D8 83		7054	RLT					
03D9 F5		7055	SEL MB1					
03DA 0400		7056	JMP 800H					
		7057	SIZECHK					
0008		7060+	SIZE SET 11					
		7061+;						
		7062+;	*****					
		7071 ;						
		7072	CODEBLK 13					
03DC		7112+	ORG 988					
03DC 28432931		7116	DB '(C)1979 INTEL'					
03E0 39373920								
03E4 494E5445								
03E8 4C								
		7117	SIZECHK					
000D		7120+	SIZE SET 13					
		7121+;						
		7122+;	*****					
		7131 ;						
		7132 ;						
		7133	R\$OURCE					
0100		7135+	PGSIZE SET ORGP00-000H ; BYTES USED ON PAGE 0					
00FD		7136+	PGSIZE SET ORGP01-100H ; BYTES USED ON PAGE 1					
0100		7137+	PGSIZE SET ORGP02-200H ; BYTES USED ON PAGE 2					
00E9		7138+	PGSIZE SET ORGP03-300H ; BYTES USED ON PAGE 3					
00FD		7139+	PGSIZE SET ORGP04-400H ; BYTES USED ON PAGE 4					
00F7		7140+	PGSIZE SET ORGP05-500H ; BYTES USED ON PAGE 5					
00FF		7141+	PGSIZE SET ORGP06-600H ; BYTES USED ON PAGE 6					
00FD		7142+	PGSIZE SET ORGP07-700H ; BYTES USED ON PAGE 7					
		7143+*	EJECT					

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTWO	THIRTY	LOC	LOC	
		7145 ;	*****		0000			
		7146 ;			0001			
		7147 ;	FILL ALL UNUSED MEMORY LOCATIONS WITH NOP OPCODES	00	+0000	00	7100	
		7148 ;			0001			
		7149 ;	*****		0002		7100	
		7150 ;			0003			
		7151 \$GEN		0	00			
		7152 ;			0004			
01FD		7160	ORG ORGPG1	0	00	+0000	00	7100
		7161	REPT (200H - ORGPG1)		0001			
		7162	DB 0	0002	0000	0000	0000	
		7163	ENDM	0003	0000	0000	0000	
01FD 00		7164+	DB 0	0	00	0000		
01FE 00		7165+	DB 0		0001	0000		
01FF 00		7166+	DB 0	0	00	+0000	00	0000
		7168 ;		0	00	+0000	00	0000
		7175 ;		0	00	+0000	00	0000
03E9		7177	ORG ORGPG3		0000			
		7178	REPT (400H - ORGPG3)		0001			
		7179	DB 0		0002			
		7180	ENDM		0003			
03E9 00		7181+	DB 0		0004			
03EA 00		7182+	DB 0		0005			
03EB 00		7183+	DB 0		0006			
03EC 00		7184+	DB 0		0007			
03ED 00		7185+	DB 0		0008			
03EE 00		7186+	DB 0		0009			
03EF 00		7187+	DB 0		0010			
03F0 00		7188+	DB 0		0011			
03F1 00		7189+	DB 0		0012			
03F2 00		7190+	DB 0		0013			
03F3 00		7191+	DB 0		0014			
03F4 00		7192+	DB 0		0015			
03F5 00		7193+	DB 0		0016			
03F6 00		7194+	DB 0		0017			
03F7 00		7195+	DB 0		0018			
03F8 00		7196+	DB 0		0019			
03F9 00		7197+	DB 0		0020			
03FA 00		7198+	DB 0		0021			
03FB 00		7199+	DB 0		0022			
03FC 00		7200+	DB 0		0023			
03FD 00		7201+	DB 0		0024			
03FE 00		7202+	DB 0		0025			
03FF 00		7203+	DB 0		0026			
		7205 ;			0027			
04FD		7207	ORG ORGPG4		0028			
		7208	REPT (500H - ORGPG4)		0029			
		7209	DB 0		0030			
		7210	ENDM		0031			
04FD 00		7211+	DB 0		0032			
04FE 00		7212+	DB 0		0033			
04FF 00		7213+	DB 0		0034			
		7215 ;			0035			
05FF		7217	ORG ORGPG5		0036			
		7218	REPT (600H - ORGPG5)		0037			

LOC	VAL	LINE	JOURNAL STATEMENT	ADDRESS	DATA	DATA
		7219	DB 0	00000000	0000	
		7220	ENDM			
05FF	00	7221+	DB 0	00000000	0000	
		7223 ;				
06FF		7225	ORG ORGPG6	00000000	0000	
		7226	REPT (700H - ORGPG6)			
		7227	DB 0	00000000	0000	
		7228	ENDM			
06FF	00	7229+	DB 0	00000000	0000	
		7231 ;				
07FD		7233	ORG ORGPG7	00000000	0000	
		7234	REPT (800H - ORGPG7)			
		7235	DB 0	00000000	0000	
		7236	ENDM			
07FD	00	7237+	DB 0	00000000	0000	
07FE	00	7238+	DB 0	00000000	0000	
07FF	00	7239+	DB 0	00000000	0000	
		7241 ;				
		7242 \$EJECT				

© Intel Corporation 1976

1

© 1996 Intel Corporation. Intel, the Intel logo, and Celeron are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

1

PSGLO 000C	RDELA' 003F	RECDO' 02CC	RECTYP 0042	KEG 0044	REORG 0005	REKOR 0198	RIN1 0011
ROTCNT 0003	ROTPAT 0002	RSOURC 0012	SCAN3 077C	SCAN5 078B	SCAN8 079D	SEGNAP 0046	SING 001A
SIZE 000D	SIZECH 0011	SMHHT 0031	SMALO 0030	STRCOM 001D	STRGOC 002C	STRMEM 0026	STRTMP 0040
STRUTL 0019	STSAVE 0500	TCRLFO 01D2	TIINT 074E	TIRET1 07A8	TOFPOL 07AC	TIYOUT 0040	TYPE 0037
UPDAD1 017C	UPDADR 0178	VERNO 0029	WBRK 0016	WDISP 06D8	WDISP1 06EE	XPCODE 0007	XPIEST 03D1
ZERO 0000							

ASSEMBLY COMPLETE, NO ERRORS

0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991	0992	0993	0994	0995	0996	0997	0998	0999
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

ISIS-II ASSEMBLER SYMBOL CROSS REFERENCE, V2.1

ASSEMBLER SYMBOL CROSS REFERENCE, V2.1																PAGE 1	
20	105#	1614	1629	1637	1650	1658	1721	1787	1806	1818	1977	1985	1999	2177	2388	2444	
	2546	2586	2719	2788	2796	2843	2857	2865	2898	2910	2928	2943	2958	2973	3007	3068	
	3096	3207	3215	3226	3234	3245	3253	3264	3272	3288	3302	3310	3323	3331	3350	3358	
	3434	3442	3453	3461	3472	3480	3491	3499	3515	3562	3583	3637	3958	3966	3986	4001	
	4016	4070	4393	4401	4592	4764	4788	4803	4819	4841	4859	4875	4962	4986	5001	5017	
	5042	5095	5167	5188	5196	5348	5360	5374	5386	5456	5464	5546	5580	5592	5600	5692	
	5704	5712	5726	5807	5956	6060	6068	6083	6157	6228	6304						
%SAVE	1235#	5460	5466	5722	5728												
%B	1280#	4413	4413	4413	4419	4459	4469	4569	4579								
%B0PNT	117#	746	754#	763	771#	780	788#	797	805#	814	822#	831	839#				
%B0R2	110#	754															
%B0R3	111#	771															
%B0R4	112#	788															
%B0R5	113#	805															
%B0R6	114#	822															
%B0R7	115#	839															
%B1PNT	126#	859	867#	880	888#	901	909#	922	930#	943	951#						
%B1R2	119#	867															
%B1R3	120#	888															
%B1R4	121#	909															
%B1R5	122#	930															
%B1R6	123#	951															
%B1R7	124#																
%BCODE	1163#	1561	1561	1561	1567	1610	1616	2390									
%BINOP	415#	1817	2387	2439	2909	3632	5179	5359	5385								
%BIT50	4217#	4411															
%BUFCN	1262#	3438	3444	3511	3517	3532	3542	3962	3968	3982	3988	4044	4054				
%BUFL	649#																
%CHARN	585#	5876															
%CHKSU	791#	3426	3426	3426	3558	3564	3571	3571	3858	3858	3858	4066	4072	4079	4079		
%CONST	104#	585	586	590	594	601	602	606	610	617	618	622	626	633	634	638	
	642	649	650	654	658	671	672	676	680	686	687	691	695	701	702	706	
	710	716	717	721	725	4217	4218	4222	4226								
%CURDI	912#																
%DEENC	617#																
%DSPTI	1037#	3092	3098														
%DSPTM	808#																
%EMPHI	1136#	5388															
%EMNLO	1127#	5362															
%EPACC	965#	2969	2975	3211	3217												
%EPPCH	1010#	2761	2777	2912	3354	3360											
%EPPCL	1001#	2734	2750	2806	2814	2819	3327	3333									
%EPPSW	974#	2792	2798	2839	2845	2894	2900	2939	2945	3249	3255	3284	3290				
%EPR0	992#	2924	2930	3268	3274	4855	4861	5060	5082								
%EPTIM	983#	2954	2960	3230	3236												
%FORM1	295#	1615	1634	1655	1688	1695	1722	1745	1780	1807	1819	1982	2000	2013	2178	2389	
	2441	2445	2547	2587	2720	2735	2742	2762	2769	2793	2811	2818	2844	2862	2877	2899	
	2911	2929	2944	2959	2974	3008	3069	3097	3212	3231	3250	3269	3289	3307	3320	3355	
	3439	3458	3477	3496	3516	3531	3563	3504	3634	3638	3807	3814	3834	3841	3963	3987	
	4002	4017	4043	4071	4263	4270	4290	4297	4322	4398	4439	4450	4550	4568	4593	4645	
	4652	4765	4789	4804	4820	4842	4860	4876	4963	4987	5002	5018	5043	5061	5068	5096	
	5168	5181	5197	5349	5361	5375	5387	5461	5504	5547	5581	5597	5616	5623	5693	5709	
	5727	5800	5957	5971	6065	6084	6158	6229	6305								
%FORM2	319#	1638	1659	1692	1702	1986	2739	2749	2766	2776	2797	2815	2825	2866	3216	3235	
	3254	3273	3311	3332	3359	3443	3462	3481	3500	3811	3821	3830	3848	3967	4267	4277	
	4294	4304	4402	4649	4659	5065	5081	5465	5601	5620	5636	5713	6069				
%FORM3	339#	1755	2023	2452	2887	3541	3651	4053	4332	4449	4468	4560	4578	5520	5981		

?FORMS	388#	1568	1576	1611	1630	1651	1684	1718	1768	1784	1803	1978	1996	2174	2250	2331
?H	2466	2484	2543	2583	2716	2731	2750	2789	2807	2840	2858	2895	2925	2940	2955	2970
?H	3004	3065	3093	3208	3227	3246	3265	3285	3303	3324	3351	3425	3435	3454	3473	3492
?H	3512	3559	3588	3803	3830	3857	3959	3983	3998	4013	4067	4259	4286	4394	4412	4509
?H	4641	4761	4785	4800	4816	4838	4856	4872	4959	4983	4998	5014	5039	5057	5092	5164
?H	5193	5345	5371	5457	5543	5577	5593	5612	5657	5675	5689	5785	5723	5884	5877	5953
?H	6061	6080	6154	6225	6301											
?H	1298#	4262	4278	4323	4333											
?HBITH	1028#	4258	4266	4271												
?HBITL	1019#	4285	4293	4296												
?HEXBU	1324#															
?HREGA	1055#															
?HREGB	1064#															
?HREGC	1073#															
?HREGD	1082#															
?HREGF	1091#															
?HREGF	1100#															
?ITMP	774#	1687	1703	1710	1710	1717	1723	1730	1730							
?KBDDBU	1208#	2332	2332	2332	2338	5615	5637	5803	5809	6153	6159					
?KEY	757#	2582	2588	2595	2595											
?KEYFL	933#															
?KEYLO	1217#	5542	5548	5658	5658	5658	5664									
?LASTK	891#	5611	5619	5624	5631	5631	5676	5676	5676							
?LDATA	740#	2810	2826	2833	2833	3633	3639	3646	3646	3652	3658	3658	4644	4660	4667	4667
?LENGT	1333#	1388	1399#	1442	1527#	1866	1876#	1936	1946#	2146	2156#	2223	2233#	2289	2299#	2356
	2366#	2507	2517#	2640	2650#	2667	2678#	3137	3147#	3383	3398#	3606	3616#	3674	3684#	3723
	3734#	3758	3770#	3909	3920#	4095	4105#	4130	4140#	4169	4179#	4206	4233#	4348	4359#	4484
	4494#	4608	4619#	4684	4694#	4713	4723#	4906	4916#	5127	5137#	5213	5224#	5253	5263#	5298
	5300#	5402	5413#	5750	5760#	5829	5839#	5898	5908#	6022	6032#	6100	6110#	6173	6184#	6248
	6258#	6323	6335#	6391	6401#	6464	6476#	6530	6541#	6601	6612#	6675	6685#	6748	6767#	6816
	6826#	6874	6884#	6935	6945#	6996	7007#	7063	7073#	7123						
?MEMHI	1154#	3886	3822	3997	4003											
?MEMLO	1145#	3833	3849	4012	4018	4640	4648	4653								
?MINDX	156#	967	971#	971	976	980#	988	985	989#	989	994	998#	998	1003	1007#	1007
	1012	1016#	1016	1021	1025#	1025	1030	1034#	1034	1039	1043#	1043	1048	1052#	1052	1057
	1061#	1061	1066	1070#	1070	1075	1079#	1079	1084	1088#	1088	1093	1097#	1097	1102	1106#
	1106	1111	1115#	1115	1120	1124#	1124	1129	1133#	1133	1138	1142#	1142	1147	1151#	1151
	1156	1160#	1160	1165	1169#	1169	1174	1178#	1178	1183	1187#	1187	1192	1196#	1196	1201
	1205#	1205	1210	1214#	1214	1219	1223#	1223	1228	1232#	1232	1237	1241#	1241	1246	1250#
	1250	1255	1259#	1259	1264	1268#	1268	1273	1277#	1277	1282	1286#	1286	1291	1295#	1295
	1300	1304#	1304	1309	1313#	1313	1317	1321#	1321	1325	1329#	1329				
?MSAVE	158#	587	603	619	635	651	673	688	703	718	742	759	776	793	810	827
	851	872	893	914	935	967	976	985	994	1003	1012	1021	1030	1039	1048	1057
	1066	1075	1084	1093	1102	1111	1120	1129	1138	1147	1156	1165	1174	1183	1192	1201
	1210	1219	1228	1237	1246	1255	1264	1273	1282	1291	1300	1309	1317	1325	1329	
?NCOLS	681#															
?NEG1	716#	2330	5674													
?NEXTP	1199#	2251	2251	2251	2257	5878	5878	5878	5884	5952	5958	5972	5962			
?NREPT	1226#	5576	5582	5596	5602											
?NUMCD	1181#	1654	1660	2173	2179	2467	2467	2467	2473	2715	2721					
?OPTIO	1190#	1633	1639	1683	1691	1696	1783	1789	1802	1808						
?OVBUF	1316#															
?OVSIZ	633#															
?PLUS1	686#	2465														
?PLUS3	701#	2249														

2R1	99#	4289	4305	4312	4312												
2KAM	103#	965	966	974	975	983	904	992	993	1001	1002	1010	1011	1019	1020	1028	
	1029	1037	1038	1046	1047	1055	1056	1064	1065	1073	1074	1082	1083	1091	1092	1100	
	1101	1109	1110	1118	1119	1127	1128	1136	1137	1145	1146	1154	1155	1163	1164	1172	
	1173	1181	1182	1190	1191	1199	1200	1208	1209	1217	1218	1226	1227	1235	1236	1244	
	1245	1253	1254	1262	1263	1271	1272	1280	1281	1289	1290	1298	1299				
2RD0	101#	740	741	745	757	758	762	774	775	779	791	792	796	808	809	813	
	825	826	830														
2RB1	102#	849	850	854	858	870	871	875	879	891	892	896	900	912	913	917	
	921	933	934	938	942												
2RELA	1244#	5688	5694	5708	5714	6064	6070	6079	6085								
2RECTY	1271#	3495	3501	3579	3585												
2KEGC	1289#	4397	4403	4440	4450	4551	4561	4508	4594								
2ROTCN	870#																
2ROTPA	849#	5505	5512	5512	5521	5527	5527										
2RSAVE	144#	591	595	607	611	623	627	639	643	655	659	677	681	692	696	707	
	711	722	726	746	763	708	797	814	831	855	859	876	880	897	901	918	
	922	939	943	4223	4227												
2SEGMA	1300#																
2SIZC	255#	1303	1437	1861	1931	2141	2218	2204	2351	2502	2635	2662	3132	3378	3601	3669	
	3718	3753	3904	4090	4125	4164	4201	4343	4479	4603	4679	4708	4901	5122	5208	5248	
	5293	5397	5745	5824	5893	6017	6095	6168	6243	6318	6386	6459	6525	6596	6670	6743	
	6811	6869	6930	6991	7058	7118											
2SMWHI	1118#	2405	2485	2485	2491	2757	2765	2770	3457	3463	3882	3810	3815	4784	4790	4982	
	4988	5182	5370	5376													
2SMALO	1109#	2730	2730	2743	2861	2867	2870	2888	3306	3312	3476	3482	3829	3837	3842	4799	
	4805	4815	4821	4837	4843	4871	4877	4997	5003	5013	5019	5038	5044	5091	5097	5192	
	5198	5344	5350	6224	6230	6300	6306										
2START	1339#	1383	1383	1391	1405#	1437	1437	1445	1533#	1861	1861	1869	1882#	1931	1931	1939	
	1957#	2141	2141	2149	2162#	2218	2218	2226	2244#	2204	2284	2292	2310#	2351	2351	2359	
	2382#	2502	2502	2510	2533#	2635	2643	2656#	2662	2662	2670	2699#	3132	3132	3132	3140	
	3173#	3378	3378	3386	3414#	3601	3601	3609	3622#	3669	3669	3677	3695#	3718	3718	3726	
	3745#	3753	3753	3761	3796#	3904	3904	3912	3951#	4090	4090	4098	4116#	4125	4125	4133	
	4151#	4164	4164	4172	4190#	4201	4201	4209	4254#	4343	4343	4351	4385#	4479	4479	4487	
	4525#	4603	4603	4611	4635#	4679	4679	4687	4700#	4708	4708	4716	4754#	4901	4901	4909	
	4952#	5122	5122	5130	5158#	5208	5208	5216	5235#	5248	5248	5256	5279#	5293	5293	5301	
	5334#	5397	5397	5405	5449#	5745	5745	5753	5791#	5824	5824	5832	5865#	5893	5893	5901	
	5939#	6017	6017	6025	6053#	6095	6095	6103	6146#	6168	6168	6176	6220#	6243	6243	6251	
	6294#	6318	6318	6326	6371#	6386	6386	6394	6437#	6459	6459	6467	6512#	6525	6525	6533	
	6582#	6596	6596	6604	6653#	6670	6670	6678	6726#	6743	6743	6751	6773#	6811	6811	6819	
	6832#	6869	6869	6877	6890#	6930	6930	6938	6951#	6991	6991	6999	7048#	7058	7058	7066	
	7114#	7118	7118	7126													
2STRIM	1253#	1981	1987	1995	2001	2014	2024										
2TYPE	1172#	1577	1577	1577	1583	1746	1756	1769	1769	1769	1775	1820	2440	2446	2453	2542	
	2548	3003	3009	3064	3070	4760	4766	4958	4964	5163	5169						
2UNARY	459#	1744	2012	2076	3530	4042	4321	4430	4457	4549	4567	5503	5970				
2VERSN	1046#																
2XPCOD	825#																
2ZEKO	671#	1559	1575	1767	2483	3424	3856	5656									
AFETCH	4704	4759#															
ASAVE	1239#	5468	5730														
ASCERR	3704	3711	3715#														
B	1284#	4415	4421	4461	4529	4571											
BCODE	1167#	1563	1569	1598	1618	2392											
BIT50	4230#	4422															
BRKEND	2462	2500#															
BRKERR	3080	3080#															

LOC	OBJ	LINE	SOURCE STATEMENT
		7243	END
USER SYMBOLS			
?A 0004	?ASAVE 0002	?B 0002	?B0PNT 0000
?B0R6 0007	?B0R7 0008	?B1PNT 0007	?B1R2 0003
?B1R7 0008	?BCODE 0002	?BINOP 0022	?BIT50 0003
?CONST 0003	?CURDI 0001	?DEBNC 0003	?DSPTI 0002
?EPPCH 0002	?EPPCL 0002	?EPPSW 0002	?CPRO 0002
?FORM4 001C	?FORM5 001E	?H 0002	?HBITH 0002
?HREGC 0002	?HREGD 0002	?HREGF 0002	?HREGG 0002
?KEYLO 0002	?LASTK 0001	?LDATA 0000	?LENGT 0000
?NCOLS 0003	?NEG1 0003	?NEXTP 0002	?NREPT 0002
?PLUS1 0003	?PLUS3 0003	?R1 0000	?RAM 0002
?REGC 0002	?ROTCN 0001	?ROTPA 0001	?RSAVE 0000
?START 03DC	?STRIM 0002	?TYPE 0002	?UNARY 002A
ASAVE 003E	ASCERR 01C9	B 0043	BCODE 0036
BRKNXT 0234	BUFCNT 0041	BUFLN 0010	BYTE11 00F2
CGOPAT 0476	CGOSS 0490	CGOTRA 0480	CGOMB 047C
CHARR 058D	CHKERR 02E1	CHKSUM 0005	C10 064D
CIN 0649	CKSMOK 02VB	CLEAR 05F1	CLRBFT 0008
CNTTBL 04A1	CNTTRA 04AA	CO1 05C5	CO2 05C8
COMGOR 0461	COMSBR 022C	COMSIZ 0003	CTAB 0023
DBLANK 05F5	DBPNT 0144	DBRK 0015	DCB 015A
DECSM1 021F	DECSMA 02F4	DELAY 04F2	DELAY1 04F5
DGR 015D	DINTRG 0169	DLST 014E	DMOD 0146
DPRMEM 015F	DREC 0151	DREL 0154	DRM 0163
DSPHI 018E	DSPLO 0194	DSPM1 0192	DSPMID 0190
DNBK 016D	ELSIF1 0007	ELSIF2 00E5	EMAH1 0033
ENDFIL 0596	ENDREC 0641	EOFREC 05AE	EPACC 0020
EPFET 07B7	EPFAS 07D0	EPFCH1 0025	EPFCL0 0024
EPRSET 0010	EPRUN 0400	EPRUN1 040A	EPRUN2 0499
EPSSTP 0004	EPSTE1 07DF	EPSTE2 07F1	EPSTEP 07D8
EXAM1 027B	EXAM2 0281	EXAM3 028A	EXAM4 0293
FDUMP2 0628	FDUMP3 0636	FDUMP4 0640	FDUMP5 0632
H02 04D5	H0DLAY 04C9	HBITHI 0027	HBITLO 0026
HFDONE 05A7	HFILE0 0572	HRECIN 0297	HRECO 0600
HREG 002E	HREGF 002F	IMPLEM 0200	INCSMR 01F2
INPAD1 00C7	INPADR 00C0	INPKEY 00EC	INVALS 0300
J10G0 0220	JTOLST 021A	JTOMOD 020F	JTOREL 0211
KBDPOL 07AF	KCLRB 000C	KEY 0003	KEYCLR 0017
KEYGO 001E	KEYLOC 003C	KEYLST 001C	KEYMOD 001F
KEYREG 001B	KEYREL 0014	KEYTRA 0019	KGORES 001D
LFEBR1 06C1	LFEBRK 06D1	LFEDM 0698	LFEINT 06A9
LFETCH 00FC	LFILL 02E9	LFILL1 02F3	LPGSEL 04E1
LSTINT 0734	LSTORE 0700	LSTPM 070C	LSTR0 072F
MADD 0024	MADD0 0025	MAIN 0029	MAIN2 0033
MAINC1 0075	MAIND 0093	MAIND1 0087	MANL 0026
MEMLO 0034	MERROR 00BC	MINC 0020	MML01 036F
MRL 002E	MRLC 0031	MRR 002F	MRRC 0030
NEXTPL 003H	NIB13 01C2	NIBIN 01B8	NIBIN2 01BA
NUNCON 0038	NXTLOC 07C8	OPTAB1 033F	OPTAB2 0346
ORGP62 0300	ORGP63 03E9	ORGP64 04FD	ORGP65 05FF
OUTUTL 0100	OVBAS 0378	OVB1 000A	OVB2 0313
OVBAS 038E	OVBUS 004E	OVLORD 036A	OVSIZ 0017
PERROR 019A	PGSIZE 00FD	PINPUT 0000	PLUS1 0001
			PLUS3 0003
			PRNT1 0117
			PRNT2 0108
			PSEGH1 0000

APPENDIX C COMMAND SUMMARY

The following is a summary of the commands implemented by the HSE-49 emulator monitor. Within each command group, tokens in each column indicate options the user has when invoking those commands.

Tokens in square brackets indicate dedicated keys on the keyboard (some keys having shared functions); angle brackets enclose hex digit strings used to specify an address or data parameter. Parameters in parentheses are optional, with the effects explained above. The notation used is as follows:

<SMA> — Starting Memory Address for block command,
<EMA> — Ending Memory Address for block command,
<LOC> — LOCAtion for individual accesses,
<DATA> — DATA byte.

Asterisks (*) indicate the default condition for each command; thus that token is optional and serves to regularize the command syntax.

Program/data entry and verification commands:

[EXAM] [PROG MEM]* <LOC> [.] [NEXT] [DATA MEM] [PREV] [REGISTER] [.] [HWRE REG] [PROG BRK] [DATA BRK]

Program/data initialization commands:

[FILL] [PROG MEM]* <SMA> [.] <EMA> [.] <DATA> [.] [DATA MEM] [REGISTER] [HWRE REG] [PROG BRK] [DATA BRK]

Intellec® development system or TTY interface commands (for transferring HEX format files):

[UPLOAD] [PROG MEM]* <SMA> [.] <EMA> [.] [DATA MEM] [REGISTER] [HWRE REG] [PROG BRK] [DATA BRK] [DNLOAD] [PROG MEM]* [.] [DATA MEM] [REGISTER] [HWRE REG] [PROG BRK] [DATA BRK]

Formatted data dump to TTY or CRT:

[LIST] [PROG MEM]* <SMA> [.] <EMA> [.] [DATA MEM] [REGISTER] [HWRE REG] [PROG BRK] [DATA BRK]

Program execution commands:

[GO] [NO BREAK]* <SMA> [.] [W/ BREAK] [.] [SING STP] [AUTO BRK] [AUTO STP] [GO/RST] [NO BREAK]* [.] [W/ BREAK] [SING STP] [AUTO BRK] [AUTO STP]

Breakpoint setting and clearing:

[SET BRK] [PROG MEM]* <LOC> [.] <LOC> ... [.] [DATA MEM] [CLR BRK] [PROG MEM]* <LOC> [.] <LOC> ... [.] [DATA MEM]

APPENDIX D ERROR MESSAGES

The following error message codes are used by the monitor software to report an operator or hardware error. Errors may be cleared by pressing [CLR/PREV] or [END/]. The format used for reporting errors is "Error—.n" where "n" is a hex digit.

Operator Errors

1. Illegal command initiator.
2. Illegal command modifier or parameter digit.
3. Illegal terminator for Examine command.
4. Illegal attempt to clear Error mode.
- 5-9. Not used.

Hardware Errors

- A. ASCII error — non-hex digit encountered in data field of hex format record.
- B. Breakpoint error. Break logic activated though breakpoints not enabled.
- C. Hex format record checksum error. Note — the checksum will not be verified if the first character of the checksum field is a question mark ("?",) rather than a hexadecimal digit. This allows object files to be patched using the ISIS text editor without the necessity of manually recomputing the checksum value.
- D. Not used.
- E. Execution processor failed to respond to a command or parameter passed to it by the master processor. EP automatically reset. EP internal status may be lost. Program memory not affected.
- F. Not used.

entire line at a time. Line printers are usually quite a bit faster than character printers, but they usually don't offer the print quality of character printers. In recent years, the "computer boom" has caused the price of printers to tumble markedly. High volume production competition and the tremendous demand for reliable print mechanisms have all contributed to the decrease in price. Because of their simplicity, line printer mechanisms have decreased in price faster than other mechanisms. Therefore, when high quality print is not needed, a line printer is a very attractive choice.

This application note describes how to control an 80 printer impact line printer with an 8085/8088. The complete software listing is included in the appendix.

The 8049 is the high performance member of the MCS-48™ microcontroller family. The Processor has all of the features of the 8048 plus twice the amount of program and data memory and an 11MHz clock speed. For details about the 8049, please refer to the MCS-48 user's manual.

II. PRINT MECHANISM DESCRIPTION

The model 820 printer is available from C. ITOH ELECTRONICS 6301 BEEHIVEN STREET, LOS ANGELES, CA 90008. This inexpensive and simple printer is ideal for applications requiring 80 columns of dot matrix alpha-numeric information.

The model 820 printer is comprised of three basic sub-assemblies: the chassis or frame, the paper feed mechanism, and the print head. The diagram in Figure 2.1 gives the physical dimensions of the basic print mechanism. The basic chassis for the printer is constructed out of four sheet metal stampings. These stampings are screwed together to form a sturdy base on which all other components of the printer are mounted.

The paper feed mechanism consists of a toothed wheel, a solenoid, a tension spring, and a "catcher." When the solenoid is activated, the arm of the solenoid pulls against the spring and drags over the toothed wheel. When the solenoid is released, the arm is pulled by the spring, but this time the arm grabs a tooth on the wheel and pulls the wheel forward which advances the paper. A "catcher," which is merely a piece of plastic held against the toothed wheel, is added to assure the paper is advanced only one "tooth" position when the solenoid is activated.

The print head is comprised of seven solenoids which are mounted in a common housing. The solenoids are actuated in a circle, but their hammers are aligned along the vertical axis. These solenoids are connected to the printer's control lines.

USING THE 8049 AS AN 80 COLUMN PRINTER CONTROLLER

I. INTRODUCTION

This Application Note details using INTEL's 8049 microcontroller as a dot matrix printer controller. Previous INTEL Application notes, i.e., AP-53 and AP-54 described using intelligent processors and peripherals to control single printer mechanisms. This Application note expands upon the theme established in these prior notes and extends the concept to include a complete bi-directional 80 column printer using a single line buffer. For convenience this application note is divided into six sections:

I. INTRODUCTION

2. PRINT MECHANISM DESCRIPTION
3. INTERFACE CIRCUITRY
4. SOFTWARE
5. CONCLUSION
6. APPENDIX

Over the last few years 80 column output devices have become somewhat of a de facto standard for business and some data processing applications. It should be mentioned that by no means is the 80 column format a "new" standard. 80 column computer terminals have been around for more than 20 years and the existence of these cards in the early days of computing is why the 80 column format is a standard today.

Many CRTs use the 80 by 24 format and to complement the number of printers use this same format. One reason for this from those history in nature, for the 80 column standard is that 80 columns of 12 pitch text on standard 8.5 inch by 11 inch paper completely fills an entire line and allows ample room for margins. The 80 column format is an aesthetically convenient format.

Printers are usually divided into impact or non-impact and a character or dot matrix printer. Impact printers actually use some form of ink or wax on the paper. More often than not, the ink is contained on a ribbon which is pulled between the paper and the paper. Non-impact printers use other than direct pressure to put characters on the paper. This type of printer is called a dot matrix printer. This type of printer is very little mechanical motion with the characters on the paper. However, because the paper is required to be treated with special care, it is not as convenient as an impact printer.

Character printers capable of printing one character at a time. A standard home typewriter is in effect a character printer. The printer must print an

Using the 8049 as an 80 Column Printer Controller

John Katusky
Applications Engineering

USING THE 8049 AS AN 80 COLUMN PRINTER CONTROLLER

I. INTRODUCTION

This Application Note details using INTEL's 8049 microcomputer as a dot matrix printer controller. Previous INTEL Application notes, (e.g. AP-27 and AP-54) described using intelligent processors and peripherals to control single printer mechanisms. This Application note expands upon the theme established in these prior notes and extends the concept to include a complete bi-directional 80 column printer using a single line buffer. For convenience this application note is divided into six sections:

1. INTRODUCTION
2. PRINT MECHANISM DESCRIPTION
3. INTERFACE CIRCUITRY
4. SOFTWARE
5. CONCLUSION
6. APPENDIX

Over the last few years 80 column output devices have become somewhat of a defacto output standard for business and some data processing applications. It should be mentioned that by no means is the 80 column format a "new" standard. 80 column computer cards have been around for more than 20 years and perhaps the existence of these cards in the early days of computers is why the 80 column format is a standard today.

Many CRT terminals use the 80 by N format and to complement this a number of printers use this same format. One reason, aside from those historic in nature, for the 80 column standard is that 80 columns of 12 pitch text on standard typewritten 8.5 inch by 11 inch paper completely fills up an entire line and allow ample room for margins. So, the 80 column format is an aesthetically convenient format.

Printers are usually divided into either impact or non-impact and a character or line oriented device. Impact printers actually use some type of "striker" to place ink on the paper. More often than not the ink is contained on a ribbon which is placed between the striker and the paper. Non-impact printers use some means other than direct pressure to place the characters on the paper. This type of printer is very fast because there is very little mechanical motion associated with placing the characters on the paper. However, because the paper is required to be treated with a special substance, it is not as convenient as an impact printer.

Character printers are capable of printing one character at a time. (Any standard home typewriter is in effect a character printer.) Line printers must print an

entire line at a time. Line printers are usually quite a bit faster than character printers, but they usually don't offer the print quality of character printers.

In recent years, the "computer boom" has caused the price of printers to tumble markedly. High volume production, competition, and the tremendous demand for reliable print mechanisms have all contributed to the decrease in price. Because of their simplicity, line printer mechanisms have decreased in price faster than other mechanisms. Therefore, when high quality print is not needed, a line printer is a very attractive choice.

This application note describes how to control an 80 column impact-line printer with an 8049/8039. The complete software listing is included in the appendix. The 8049 is the high-performance member of the MCS-48™ microcontroller family. The Processor has all of the features of the 8048 plus twice the amount of program and data memory and an 11MHz clock speed. For details about the 8049, please refer to the MCS-48 user's manual.

II. PRINT MECHANISM DESCRIPTION

The model 820 printer is available from C. ITOH ELECTRONICS (5301 BEETHOVEN STREET, LOS ANGELES, CA 90066). This inexpensive and simple printer is ideal for applications requiring 80 columns of dot matrix alpha-numeric information.

The model 820 printer is comprised of three basic sub-assemblies; the chassis or frame, the paper feed mechanism, and the print head. The diagram in Figure 2.1 gives the physical dimensions of the basic print mechanism. The basic chassis for the printer is constructed out of four sheet metal stampings. These stampings are screwed together to form a sturdy base on which all other components of the printer are mounted.

The paper feed mechanism consists of a toothed wheel, a solenoid, a tension spring, and a "catcher." When the solenoid is activated, the arm of the solenoid pulls against the spring and drags over the toothed wheel. When the solenoid is released, its arm is pulled by the spring, but this time the arm grabs a tooth on the wheel and pulls the wheel forward which advances the paper. A "catcher," which is merely a piece of plastic held against the toothed wheel, is added to assure that the paper is advanced only one "tooth" position each time the solenoid is activated.

The print head is comprised of seven solenoids which are mounted in a common housing. The solenoids are physically mounted in a circle, but their hammers are positioned linearly along the vertical axis. These seven vertically positioned hammers are the strikers that actually do the printing.

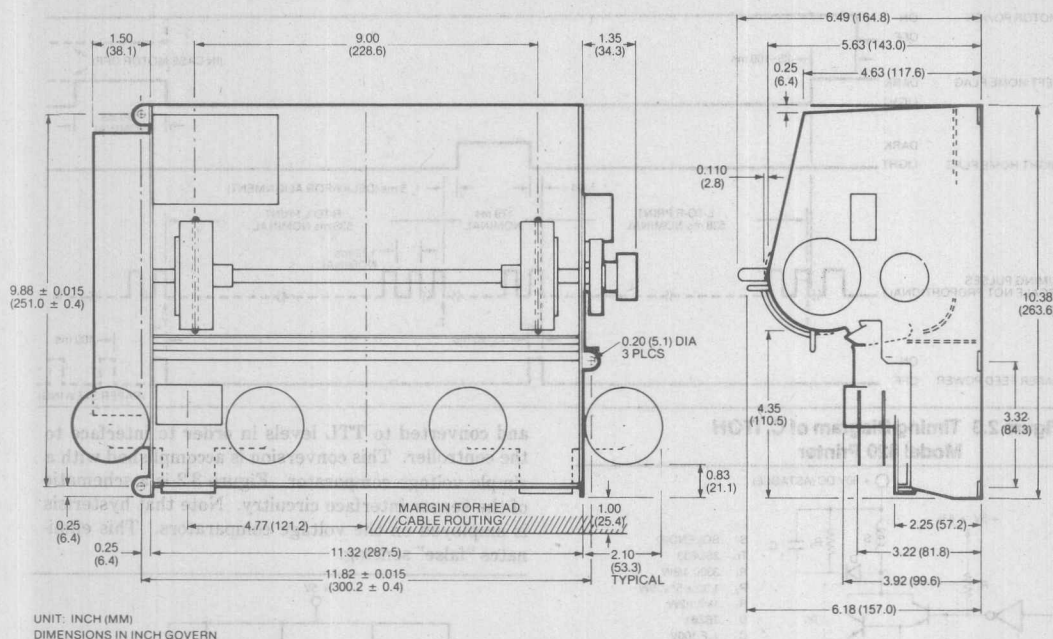


Figure 2.1 Physical Dimensions of C. ITOH Model 820 Printer

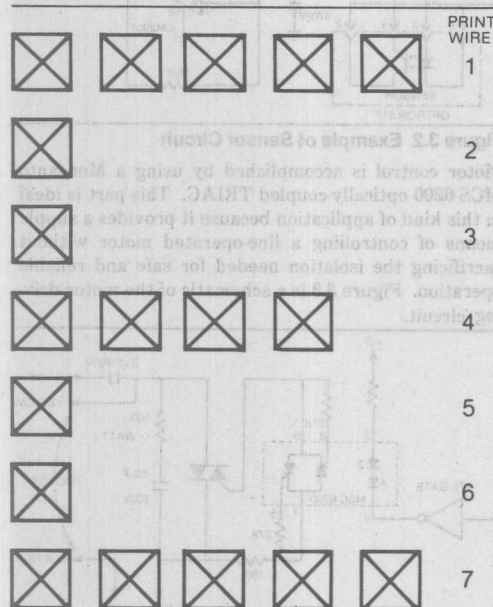


Figure 2.2 "Formation" of a Character by a Dot Matrix Printer

A motor, mounted toward the back of the print mechanism, drives a rubber toothed belt which turns a roller guide. A motor turns a guide that moves the print head from right to left and left to right. By properly timing the current flow through the solenoids while the print head is moving across the paper, characters can be formed. Figure 2.2 illustrates how the dot matrix printer "forms" its characters.

The timing pulses for the print head mechanism are generated by an opto-electronic sensor. This sensor, located on the left side plate of the printer, informs the print controller when to apply current to the print head mechanism. This "on-board timing wheel" assures that all characters will be properly spaced and that they will all be "in-line" in a vertical sense.

The print mechanism is also equipped with two additional sensors. These are the left home position sensor, located near the left front of the mechanism, and the right home position sensor, located near the right front of the print mechanism. These sensors simply tell the controller when the print head is in either the left or right home position. A complete timing chart for the printer is shown in Figure 2.3.

III. INTERFACE CIRCUITRY

The manual supplied with the printer recommends some specific interface circuitry. For the most part the circuitry used in this Application Note followed these suggestions. The circuitry needed to drive the print head solenoid is shown in Figure 3.1. This same

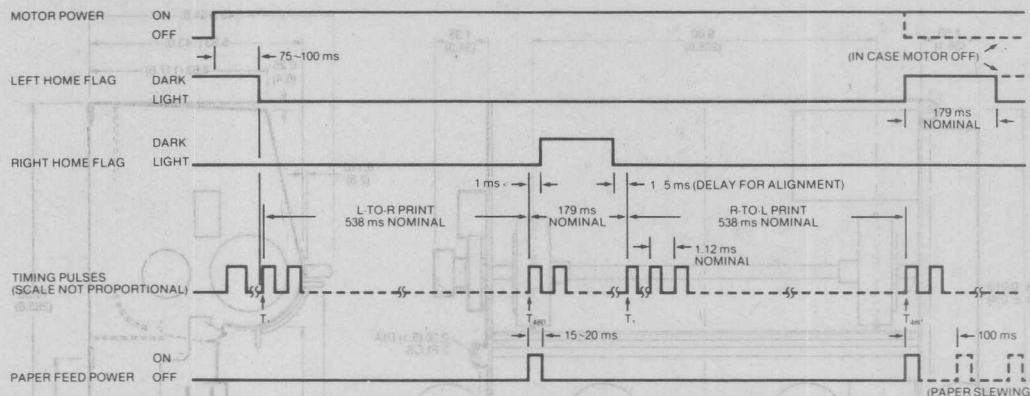


Figure 2.3 Timing Diagram of C. ITOH Model 820 Printer

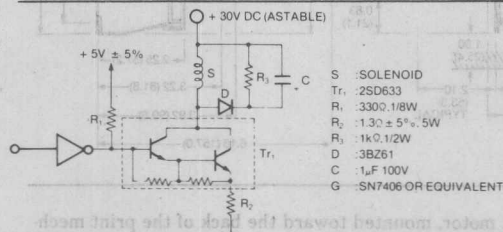


Figure 3.1 Solenoid Drive Circuit (Eliminate R2 for Line Feed Solenoid)

circuit is used to drive the line feed solenoid except that the current limiting resistor R2 is eliminated. This resistor is not needed because the line feed solenoid is physically much larger than the print head solenoids and can tolerate much higher levels of current.

The print head drivers are connected to an 8212 latch. The latch is interfaced to the BUS PORT on the 8049 and is enabled whenever the WR pin and the BIT 4 of PORT 1 are coincidentally low. The line feed driver is connected to PORT 1 BIT 1 of the 8049.

Note that the driver is simply a Darlington transistor that is driven by an open collector TTL gate. Resistor R2 is the current limiting resistor and diode D, capacitor C, and resistor R3 are used to "dampen" the inductive spike that occurs when driving solenoid S. This circuit is repeated for each of the seven solenoids in the print head. It should be mentioned that, although the type of Darlington transistor needed to drive the print head is not critical, a collector current rating of at least 5 amps and a breakdown voltage (Vceo) of at least 100 volts is needed. Transistors that do not meet these requirements will be damaged by the inductive kickback of the solenoids.

As mentioned in Section 2, the printer provides some sensor interface signals that are derived via three optoelectronic sensors. These signals must be amplified

and converted to TTL levels in order to interface to the controller. This conversion is accomplished with a simple voltage comparator. Figure 3.2 is a schematic of the sensor interface circuitry. Note that hysteresis is employed on the voltage comparators. This eliminates "false" sensing.

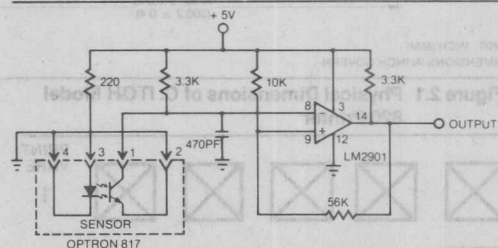


Figure 3.2 Example of Sensor Circuit

Motor control is accomplished by using a Monsanto MCS-6200 optically-coupled TRIAC. This part is ideal in this kind of application because it provides a simple means of controlling a line-operated motor without sacrificing the isolation needed for safe and reliable operation. Figure 3.3 is a schematic of the motor driving circuit.

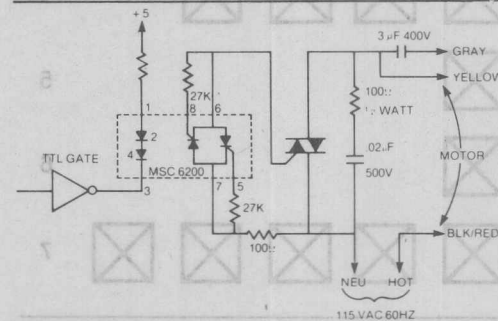


Figure 3.3 Motor Driving Circuit

To interface 8049 to the outside world one 8212 latch was used. This latch was connected to the BUS PORT and is enabled by an INS or MOVX instruction coincident with BIT 4 of PORT 1 being in a logical zero state. In this configuration, the 8212 was used to hold the data until read by the 8049. The connection of the 8212 to the 8049 is shown in Figure 3.4 and the parallel port timing diagram is shown in Figure 3.5. The 8212 parallel port was connected to the LINE PRINTER OUTPUT of an INTELLEC MICROCOMPUTER DEVELOPMENT SYSTEM.

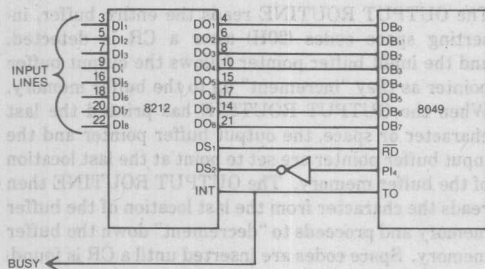


Figure 3.4 Connection of the 8212 Input Port to the 8049

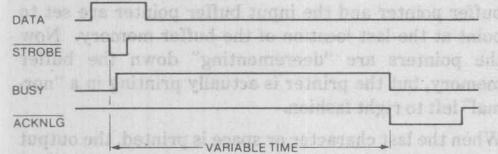


Figure 3.5 Parallel Port Timing

IV. SOFTWARE

As mentioned in Section 2, the bulk of the timing needed to control the printer is actually generated by the printer itself. Therefore, all the software must do is harness these timing signals and turn on and off the right solenoids at the right time.

To make things easy, the software needed to drive the printer is broken into four separate routines. These are:

1. INITIALIZATION ROUTINE
2. INPUT ROUTINE
3. OUTPUT ROUTINE
4. LOOKUP ROUTINE

The INITIALIZATION ROUTINE turns the motor on and checks the opto-electronic sensors. If a failure is found, the routine turns off the motor and loops on itself. This insures that the print mechanism is cycled properly before characters are accepted for printing.

This routine also initializes all of the variables used by the printer.

The INPUT ROUTINE reads the characters that are present in the 8212 input port and writes them into the 8049's buffer memory. The routine then checks the characters to see if a CARRIAGE RETURN (ASCII OCH) has been transmitted. If a CR is detected, the input routine automatically inserts a LINE FEED as the next character. When the input routine detects a LINE FEED, it stops reading characters and sets the direction bits and the print bit in the status register. This action evokes the OUTPUT ROUTINE. A detailed flowchart of the INPUT ROUTINE is shown in Figure 4.1.

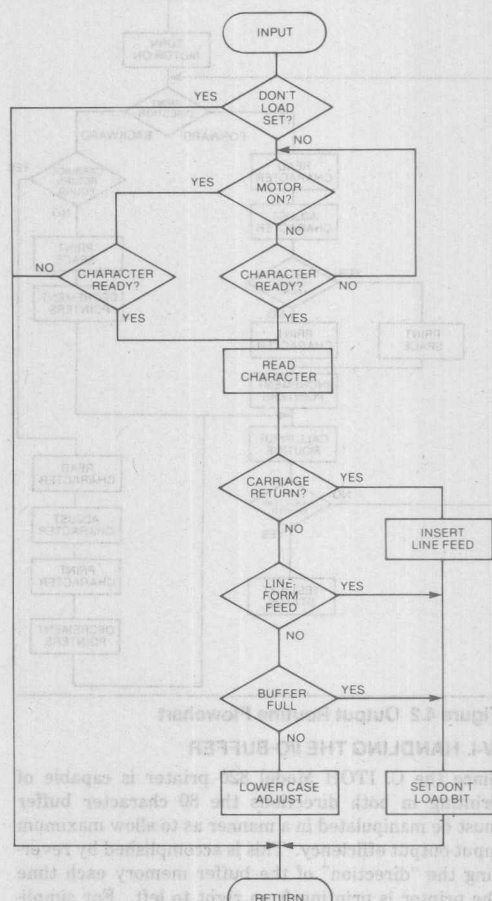


Figure 4.1 Input Routine Flowchart

The OUTPUT ROUTINE initializes both the input and output buffer pointers and then reads the characters from the 8049's buffer memory. After a character is read the OUTPUT ROUTINE calls the LOOKUP ROUTINE which reads the proper bit pattern to form that character. This bit pattern is then used to strobe the solenoids. After each character is printed, the OUTPUT ROUTINE calls the INPUT ROUTINE and another character is placed into the buffer memory. This type of operation guarantees that the input buffer cannot "overflow" the output buffer. A flowchart of the OUTPUT ROUTINE is shown in Figure 4.2.

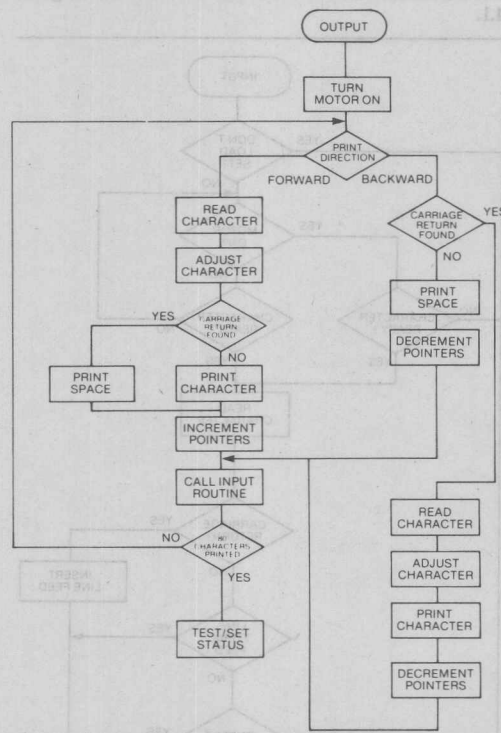


Figure 4.2 Output Routine Flowchart

IV-I. HANDLING THE I/O BUFFER

Since the C. ITOH Model 820 printer is capable of printing in both directions the 80 character buffer must be manipulated in a manner as to allow maximum input-output efficiency. This is accomplished by reversing the "direction" of the buffer memory each time the printer is printing from right to left. For simplicity, if it is assumed that the buffer is only five bytes long, Figure 4.3 can be used to help explain the buffer operation.

Initially the input buffer pointer is loaded with the address of the first location in the buffer memory. As characters are read, the input buffer pointer increments and fills the buffer memory as shown in Figure 4.3(b) through 4.3(f). When a CARRIAGE RETURN-LINE FEED (CRLF) is encountered the input buffer pointer and the output buffer pointer are reset back to the first location. The OUTPUT ROUTINE then reads the character from the first location in the buffer memory, increments the output buffer pointer and calls the INPUT ROUTINE, which reads another character from the parallel input port.

The OUTPUT ROUTINE reads the entire buffer, inserting space codes (20H) after a CR is detected, and the input buffer pointer follows the output buffer pointer as they "increment" up to the buffer memory. When the OUTPUT ROUTINE has printed the last character or space, the output buffer pointer and the input buffer pointer are set to point at the last location of the buffer memory. The OUTPUT ROUTINE then reads the character from the last location of the buffer memory and proceeds to "decrement" down the buffer memory. Space codes are inserted until a CR is found. Figure 4.3(1) to 4.3(0).

The input buffer pointer follows the output buffer pointer just as in the previous case. When the last, or in this case the first character is printed, the output buffer pointer and the input buffer pointer are set to point at the last location of the buffer memory. Now the pointers are "decrementing" down the buffer memory, but the printer is actually printing in a "normal" left to right fashion.

When the last character or space is printed, the output buffer and the input buffer pointer are set to the first location of the buffer memory and printing takes place in a reverse or right to left manner. After this line is printed, the print head and both buffer pointers are in the same position as they were initially. So, four lines must be printed before the buffer pointers and the print head complete a cycle. Each of these situations is handled separately by four different sub-routines: CASE0, CASE1, CASE2, and CASE3.

IV-II. TIMING

All critical timing for the printer controller came from two basic sources; the timing sensors on the printer and the internal eight-bit timer of the 8049.

The internal timer of the 8049 was used to control the length of time the solenoids were fired (600 microseconds) and was also used as a "one-shot" to align the printer. This alignment is needed to make the "backward" printing line up vertically with the normal or forward printing. The "one-shot" is used to measure the time from the last column of the last character position until the right sensor flag is covered.

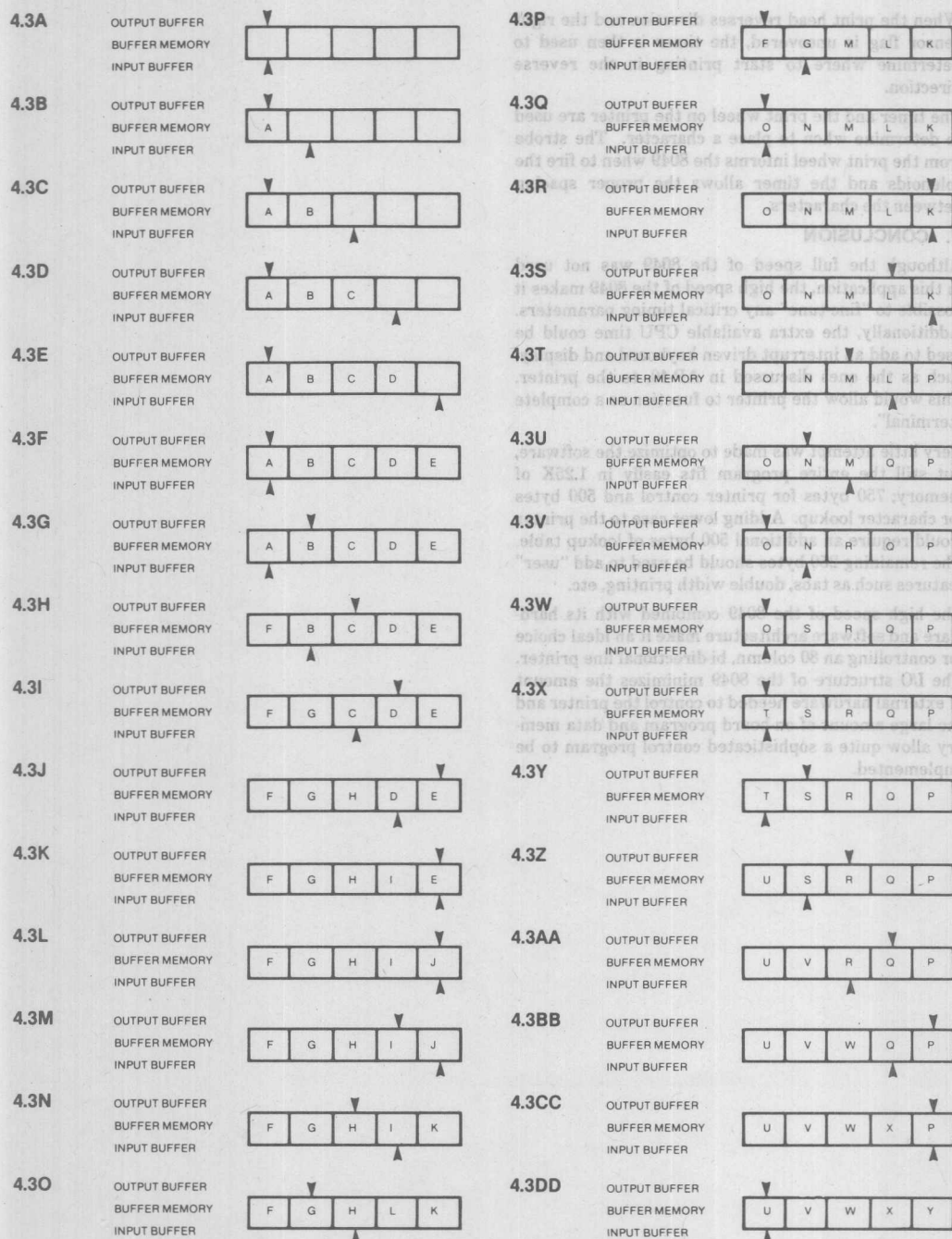


Figure 4.3 I/O Buffer Handler

When the print head reverses direction and the right sensor flag is uncovered, the timer is then used to determine where to start printing in the reverse direction.

The timer and the print wheel on the printer are used to determine when to place a character. The strobe from the print wheel informs the 8049 when to fire the solenoids and the timer allows the proper spacing between the characters.

V. CONCLUSION

Although the full speed of the 8049 was not used in this application, the high speed of the 8049 makes it possible to "fine-tune" any critical timing parameters. Additionally, the extra available CPU time could be used to add an interrupt driven keyboard and display, such as the ones discussed in AP-40, to the printer. This would allow the printer to function as a complete "terminal".

Very little attempt was made to optimize the software, but still the entire program fits easily in 1.25K of memory; 750 bytes for printer control and 500 bytes for character lookup. Adding lower case to the printer would require an additional 500 bytes of lookup table. The remaining 250 bytes should be used to add "user" features such as tabs, double width printing, etc.

The high speed of the 8049 combined with its hardware and software architecture make it an ideal choice for controlling an 80 column, bi-directional line printer. The I/O structure of the 8049 minimizes the amount of external hardware needed to control the printer and the large amount of on-board program and data memory allow quite a sophisticated control program to be implemented.

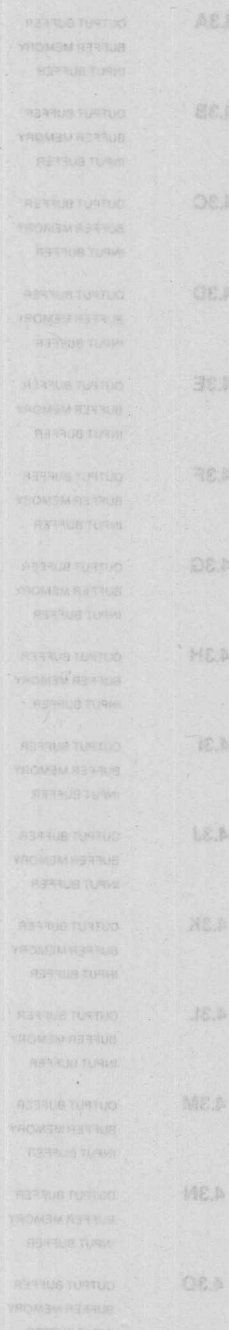
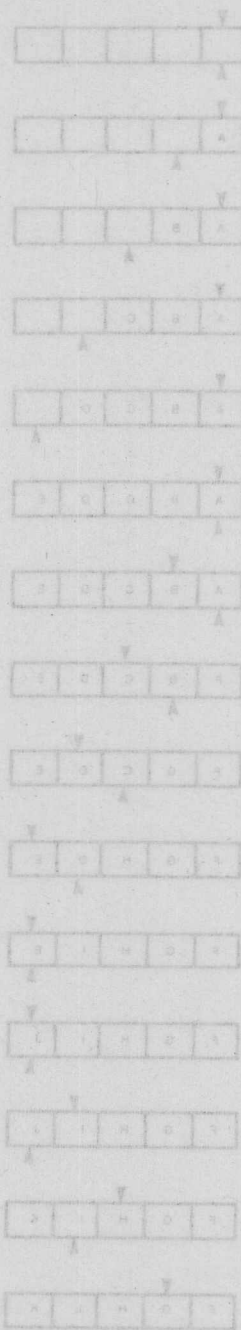
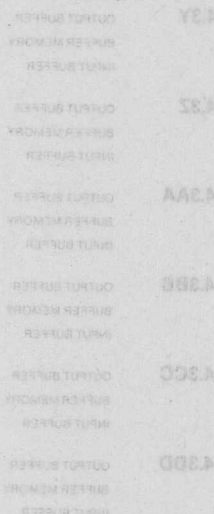
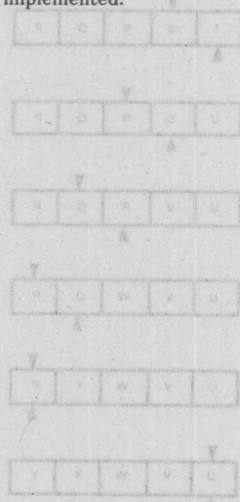
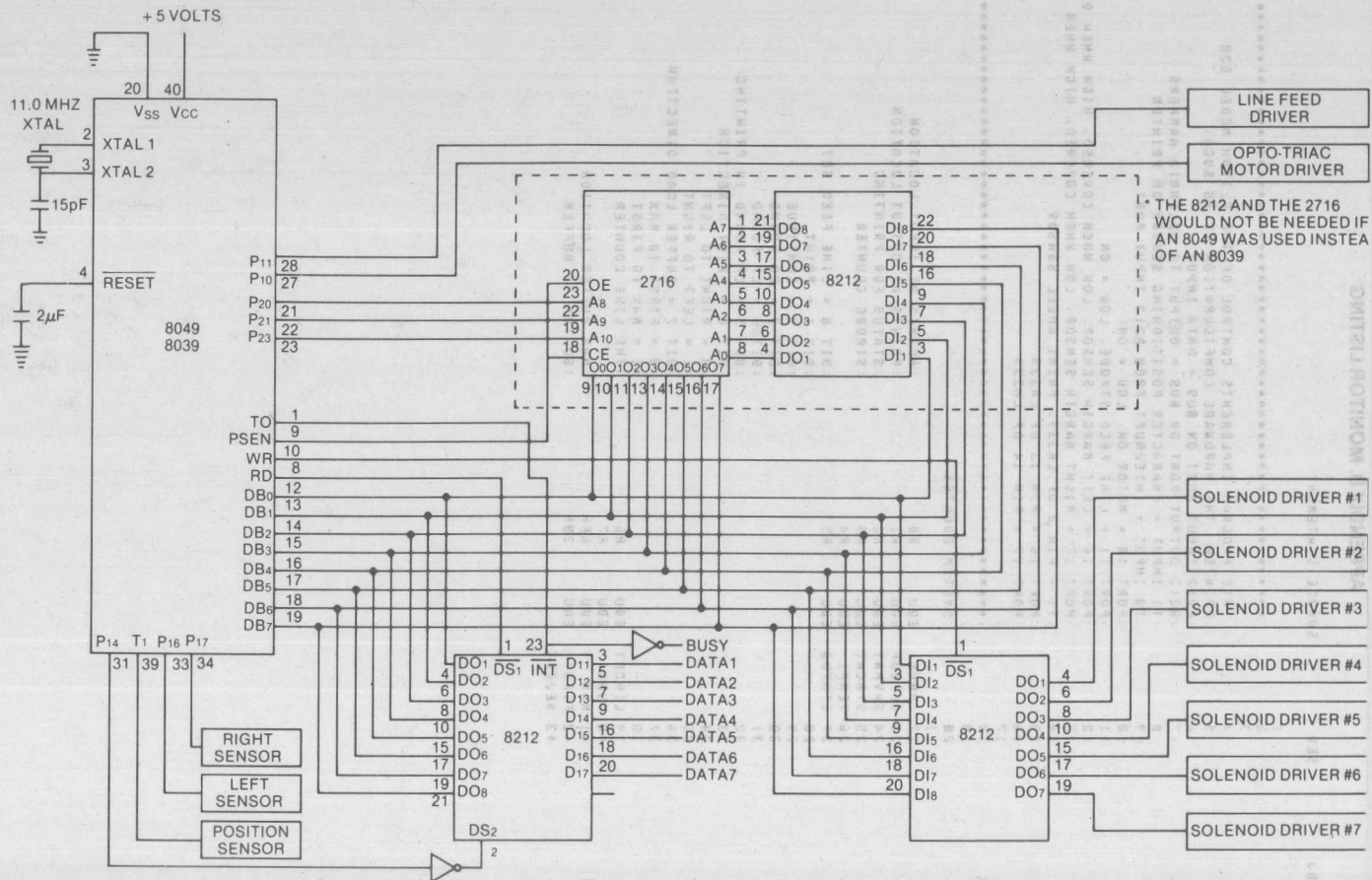


Figure 4-3: I/O Buffer Handler

APPENDIX A. SCHEMATIC DIAGRAM



LOC OBJ

SEQ

SOURCE STATEMENT

0000
0001
0002
0003
0004
0005

0006
0007
000F
0020

```

1 .....
2 .....
3 .....
4 .....
5 THIS PROGRAM IMPLEMENTS CONTROL OF THE C-170N MODEL 82B
6 PRINTER THE HARDWARE CONFIGURATION IS AS SUCH:
7 8212 INPUT PORT ON BUS = DATA INPUT
8 8212 OUTPUT PORT ON BUS = OUTPUT TO SOLENOID HAMMERS
9 T1 INPUT = CHARACTER POSITIONING SENSOR ON PRINTER
10 T0 INPUT = INTERRUPT FROM 8212 INPUT PORT
11 PORT 10 = MOTOR ON, LOW = ON
12 PORT 11 = LINE FEED STROBE, LOW = ON
13 PORT 16 = LEFT MARGIN SENSOR, LOW WHEN COVERED, HIGH WHEN OPEN
14 PORT 17 = RIGHT MARGIN SENSOR, LOW WHEN COVERED, HIGH WHEN OPEN
15 T1 = PIN 2 OF LM339, PRINT WHEEL SENSOR
16 PORT 16 = PIN 13 OF LM339
17 PORT 17 = PIN 14 OF LM339
18 .....
19 .....
20 SYSTEM EQUATES
21 .....
22 INBUF EQU R0 POINTS AT INPUT LOCATION
23 OUTBUF EQU R1 POINTS AT OUTPUT LOCATION
24 SAVPNT EQU R2 STATUS FOR PRINTING
25 STBCNT EQU R3 STROBE COUNTER
26 TENP1 EQU R4
27 STATUS EQU R5
28
29
30
31
32
33
34
35
36
37
38
39 LINCNT EQU R6
40 JUNK1 EQU R7
41 MAX EQU 6FH
42 FIRST EQU 2BH
43 $EJECT

```

BIT 0 = LINE FEED SET
 BIT 1 = PRINT
 BIT 2 = CONTINUE
 BIT 3 = CR FOUND
 BIT 4 = LF FOUND
 BIT 5 = LF FOUND IN PRINTING
 BIT 6 = PRINT DIRECTION
 0 = RIGHT TO LEFT
 1 = LEFT TO RIGHT
 BIT 7 = BUFFER LOAD DIRECTION
 0 = FIRST TO MAX
 1 = MAX TO FIRST
 THE LINE COUNTER
 MAX BUFFER LOCATION
 BOTTOM OF BUFFER

LOC	OBJ	SEQ	SOURCE STATEMENT	THIRSTATE 330002	330	LOC 307
		44	SETZ BANC BNT T201	00000000	0000	10 1000
0000		45	ORIG ORG 000 000000	00000000	0000	10 1000
		46	SETZ BANC BNT T201	00000000	0000	10 1000
		47	JUMP OVER THE INTERRUPT LOCATIONS	00000000	0000	10 1000
		48	SETZ BANC BNT T201	00000000	0000	10 1000
3000 15		49	DISZ 001 T201	00000000	0000	10 1000
0001 3400		50	UNP3402 BEGIN	00000000	0000	10 1000
		51	SETZ BANC BNT T201	00000000	0000	10 1000
000A		52	ORG BAH 001	00000000	0000	10 1000
		53	SETZ BANC BNT T201	00000000	0000	10 1000
		54	SETZ BANC BNT T201	00000000	0000	10 1000
		55	SETZ BANC BNT T201	00000000	0000	10 1000
		56	LOOP UNTIL THE BUFFER FILLS UP	00000000	0000	10 1000
		57	SETZ BANC BNT T201	00000000	0000	10 1000
000A FD		58	PRT: MOV BANC A STATUS	00000000	0000	10 1000
000B 3211		59	NO BNT B1102 LPRNT	00000000	0000	10 1000
000D 3400		60	CALL LDBUF	00000000	0000	10 1000
000F 040A		61	JMP PRT	00000000	0000	10 1000
		62		00000000	0000	10 1000
		63	THIS ROUTINE PRINTS A LINE	00000000	0000	10 1000
		64	IT FIRST SAVES THE STATUS	00000000	0000	10 1000
		65	AND THEN DETERMINES WHICH DIRECTION TO PRINT	00000000	0000	10 1000
		66	AND HOW TO MANIPULATE THE BUFFER	00000000	0000	10 1000
		67	SETZ BANC BNT T201	00000000	0000	10 1000
0011 3409		68	LPRNT: JMP BNT SWCHK	00000000	0000	10 1000
0013 F224		69	LPRNT: JB7 CASE23	00000000	0000	10 1000
0015 0417		70	JMP BNT CASE01	00000000	0000	10 1000
		71	SETZ BANC BNT T201	00000000	0000	10 1000
		72	CASE01: LOADING THE BUFFER FROM FIRST TO MAX	00000000	0000	10 1000
		73	SETZ BANC BNT T201	00000000	0000	10 1000
0017 0920		74	CASE01: MOV BANC OUTBUF, #FIRST	00000000	0000	10 1000
0019 0820		75	MOV INBUF, #FIRST	00000000	0000	10 1000
001B FA		76	MOV BANC A SAVPNT	00000000	0000	10 1000
001C 940C		77	CALL MOTON	00000000	0000	10 1000
001E D252		78	JB6 CASE1	00000000	0000	10 1000
0020 94B3		79	CALL BNT PRTBK	00000000	0000	10 1000
0022 0431		80	JMP BNT CASE01	00000000	0000	10 1000
		81	SETZ BANC BNT T201	00000000	0000	10 1000
		82	CASE23: LOADING BUFFER FROM MAX TO FIRST	00000000	0000	10 1000
		83	SETZ BANC BNT T201	00000000	0000	10 1000
0024 096F		84	CASE23: MOV OUTBUF, #MAX	00000000	0000	10 1000
0026 086F		85	MOV INBUF, #MAX	00000000	0000	10 1000
0028 FA		86	MOV A SAVPNT	00000000	0000	10 1000
0029 940C		87	CALL MOTON	00000000	0000	10 1000
002B D2C2		88	JB6 CASE3	00000000	0000	10 1000
002D 94B3		89	CALL PRTBK	00000000	0000	10 1000
002F 043D		90	JMP CASE2	00000000	0000	10 1000
		91		00000000	0000	10 1000
		92	EJECT	00000000	0000	10 1000

LDC OBJ	SEQ	SOURCE STATEMENT	THESE ARE COMMENTS	LOC	LOC
	134	;			
	135	THIS ROUTINE CALLS THE LINE FEED			
	136	;			
0071 9478	137	DDLF: CALL LINEFD	STROBE LINE FEED SOLENOID		
0073 848A	138	JMP PRNT	GO BACK TO THE PRINT ROUTINE		
	139	;			
	140	THIS ROUTINE COMPLETES A LINE WHEN THE PRINT			
	141	HEAD IS MOVING LEFT TO RIGHT			
	142	;			
0075 27	143	WATCH: CLR INT AOUT	ZERO ACC		
0076 62	144	MOV INT A	ZERO TIMER		
0077 55	145	STRT T	START THE TIMER		
0078 348B	146	CALL LDBUF	READ THE LAST CHARACTER		
007A 89	147	LDOPW: IN A, P1	EXAMIN PORT ONE		
007B F27A	148	JNB7H1 LOOPW	CHECK RIGHT HAND SENSOR		
007D 65	149	STOP TCHT	STOP THE TIMER		
007E FD	150	MOV A, STATUS	GET THE STATUS		
007F 5285	151	JNB7H1 JNB7H1	JUMP IF CONTINUE IS SET		
0081 94DF	152	CALL MOTOF	TURN MOTOR OFF		
0083 53FD	153	ANL A, #BFH	RESET BIT ONE		
0085 53FB	154	OVRI: ANL A, #BFH	RESET CONTINUE BIT		
0087 AD	155	MOV STATUS, A	RESTORE STATUS		
0088 FA	156	MOV A, SAVPNT	GET THE SAVED STATUS		
0089 B271	157	JNB7H1 JNB7H1	ADD A LINE FEED IF BIT IS SET		
008B 848A	158	JMP PRNT	GO BACK TO PRINT ROUTINE		
	159	;			
	160	;			
	161	CASE 2: PRINTING RIGHT TO LEFT, LOADING BUFFER FROM			
	162	;			
	163	;			
	164	;			
008D F1	165	CASE2: MOV A, #OUTBUF	GET THE CHARACTER		
008E 3491	166	CALL FXPRNT	ADJUST FOR PRINTING		
0090 B12B	167	MOV A, #OUTBUF, #2BH	PUT A SPACE IN BUFFER RAM		
0092 F29E	168	JNB7H1 JNB7H1	FIND A CR YET		
0094 9472	169	CALL DECTST	CHECK THE BUFFER		
0096 C6AE	170	JZ WATCHD	IF ZERO WAIT FOR SENSOR FLAG		
0098 BF2B	171	MOV JUNK1, #2BH	PUT SPACE IN TEMP LOCATION		
009A 9463	172	CALL GTPRNT	GO PRINT SPACE		
009C 848D	173	JMP CASE2	LOOP		
009E BF2B	174	FDCR: MOV JUNK1, #2BH	GET A SPACE		
00A0 9463	175	FDCR1: CALL GTPRNT	GO PRINT THE CHARACTER		
00A2 9472	176	CALL DECTST	CHECK THE BUFFER		
00A4 C6AE	177	JZ WATCHD	LEAVE IF DONE		
00A6 F1	178	MOV A, #OUTBUF	GET A CHARACTER		
00A7 3491	179	CALL FXPRNT	ADJUST THE CHARACTER FOR PRINTING		
00A9 AF	180	MOV JUNK1, A	SAVE IT		
00AA B12B	181	MOV #OUTBUF, #2BH	PUT A SPACE WHERE THE CHARACTER WAS		
00AC 848B	182	JMP FDCR1	LOOP		
	183	EJECT			

LOC	OBJ	SEQ	SOURCE STATEMENT	TRANS	MTZ	339002	032	480	307
B100		222	ORG 100H						
		223							
B100 B9		224	LDBUF: IN A,P1						
B101 B21C		225	JB5 LMODE						
B103 12B7		226	JB5 ARND						
B105 89B1		227	ORL P1,#B1H						
B107 92BF		228	ARND: JB4 MOFF						
B109 FE		229	MOV A,LINCHT						
B10A 438B		230	ORL A,#BBH						
B10C AE		231	MOV LINCHT,A						
B10D 23FF		232	MOV A,#BFFH						
B10F 721A		233	MOFF: JB3 NOLF						
B111 9478		234	CALL LINEFD						
B113 B9		235	BUTLOP: IN A,P1						
B114 721A		236	JB3 NOLF						
B116 921A		237	JB4 NOLF						
B118 2413		238	JMP BUTLOP						
B11A 24BB		239	NOLF: JMP LDBUF						
		240							
		241							
		242							
		243							
B11C 261F		243	LMODE: JNB CHAR						
B11E 83		244	RET						
		245							
		246							
		247							
B11F FD		248	CHAR: MOV A,STATUS						
B120 5249		249	JB2 ARNDJP						
B122 9249		250	JB4 ARNDJP						
B124 724A		251	JB3 LFCRCK						
B126 94D6		252	CALL GTCAR						
B128 3461		253	GOOD: CALL FXCHAR						
B12A AB		254	MOV R1NBUF,A						
B12B FD		255	MOV A,STATUS						
B12C F239		256	JB7 SUB1						
B12E 18		257	INC INBUF						
B12F 237B		258	MOV A,#MAX+1						
B131 D8		259	XRL A,INBUF						
B132 9649		260	JNZ ARNDJP						
B134 F8		261	MOV A,INBUF						
B135 B7		262	DEC A						
B136 AB		263	MOV INBUF,A						
B137 2449		264	JMP ARNDJP						
B139 F8		265	SUB1: MOV A,INBUF						
B13A B7		266	DEC A						
B13B AB		267	MOV INBUF,A						
B13C 231F		268	MOV A,#FIRST-1						
B13E D8		269	XRL A,INBUF						
B13F 9649		270	JNZ ARNDJP						
B141 18		271	INC INBUF						
B142 2449		272	JMP ARNDJP						
B144 FD		273	GETSTA: MOV A,STATUS						
B145 1249		274	JB5 ARNDJP						
B147 925B		275	ARND: JB4 STBIT1						
B149 83		276	ARNDJP: RET						
		277							
		278							
		279							
B14A 94D6		280	LFCRCK: CALL GTCAR						
B14C 238A		281	MOV A,#BAH						
B14E 2428		282	JMP GOOD						
		283							
		284							
		285							
B15B FD		286	STBIT1: MOV A,STATUS						
B151 3259		287	JB1 STPRNT						
B153 43B2		288	ORL A,#B2H						
B155 B34B		289	ADD A,#4BH						
B157 AD		290	MOV STATUS,A						
B158 83		291	RET						
B159 526B		292	STPRNT: JB2 BYEBYE						
B15B 43B4		293	ORL A,#BAH						
B15D B34B		294	ADD A,#4BH						
B15F AD		295	MOV STATUS,A						
B16B 83		296	BYEBYE: RET						
		297							

LOC	OBJ	SEQ	SOURCE STATEMENT
		290	;THIS ROUTINE "CONVERTS" LOWER CASE LETTERS TO
		299	;UPPER CASE
		300	
B161 97		301	FXCHAR: CLR C ;CLEAR THE CARRY
B162 537F		302	AHL A,07FH ;STRIP MSB
B164 AF		303	MOV A,JUNK1.A ;SAVE ACC
B165 B3AB		304	ADD A,00ABH ;SEE IF NUMBER IS 6BH
B167 E670		305	JNC FINE ;IF CARRY ISN'T SET, JUMP
B169 FF		306	MOV A,JUNK1 ;GET ACC BACK
B16A 37		307	CPL A ;SUBTRACT 2BH FROM THE ACC
B16B B320		308	ADD A,02BH
B16D 37		309	CPL A
B16E 2474		310	JMP FIXDUN ;JUMP TO TEST CR LF
B170 37		311	FINE: CPL A ;NOW SUBTRACT ABH FROM ACC
B171 B3AB		312	ADD A,00ABH
B173 37		313	CPL A
B174 AF		314	FIXDUN: MOV JUNK1.A ;SAVE A
B175 D3BD		315	XRL A,00DH ;IS CHARACTER A CR
B177 967F		316	JNZ LFTEST ;IF IT IS NOT TEST LF
B179 FD		317	MOV A,STATUS ;GET THE STATUS
B17A 4300		318	ORL A,000H ;SET BIT 3
B17C AD		319	MOV STATUS,A ;RESTORE THE STATUS
B17D 240F		320	JMP FIXFIN ;LEAVE
B17F FF		321	LFTEST: MOV A,JUNK1 ;GET CHARACTER BACK
B180 D3BA		322	XRL A,00AH ;IS IT A LF
B182 C609		323	JZ FIXUP ;IF ITS NOT, WE ARE DONE
B184 FF		324	MOV JUNK1 ;GET THE CHARACTER BACK
B185 D3BC		325	XRL A,00CH ;IS IT A FORM FEED
B187 960F		326	JNZ FIXFIN ;IF NOT FORM FEED, JUMP
B189 FD		327	FIXUP: MOV A,STATUS ;GET THE STATUS
B18A 4310		328	ORL A,010H ;SET BIT 4
B18C AD		329	MOV STATUS,A ;RETURN THE STATUS
B18D 3450		330	CALL STBIT ;SET THE STATUS
B18F FF		331	FIXFIN: MOV JUNK1 ;GET THE CHARACTER
LOC OBJ		SEQ	SOURCE STATEMENT
B190 03		332	RET ;EXIT FIXCHAR
		333	
		334	;THIS ROUTINE RECOGNIZES A LF, FF, AND CR
		335	;DURING THE PRINT OPERATION
		336	;IT ALSO FORCES A SPACE IF A CHARACTER FOUND
		337	;IN THE BUFFER IS NOT IN THE LOOKUP TABLE
		338	
B191 AF		339	FXPRNT: MOV JUNK1.A ;SAVE ACC
B192 D3BC		340	XRL A,00CH ;FORM FEED
B194 C602		341	JZ JZ ;GO SET FORM FEED
B196 FF		342	MOV JUNK1 ;RESTORE CHARACTER
B197 D3BD		343	XRL A,00DH ;SEE IF IT IS A CR
B199 C6A8		344	JZ CRFIX ;LEAVE IF IT IS
B19B FF		345	MOV JUNK1 ;GET ACC BACK
B19C D3BA		346	XRL A,00AH ;SEE IF IT IS A LF
B19E C6A8		347	JZ LFFIX ;LEAVE IF IT IS
B1A0 FF		348	MOV JUNK1 ;GET CHARACTER BACK
B1A1 53EB		349	AHL A,00EBH ;SEE IF IT IS A CHARACTER
B1A3 96BD		350	JNZ ISCHAR ;IF IT IS JUMP
B1A5 2320		351	MOV A,02BH ;PUT A SPACE IN ACC
B1A7 03		352	RET ;EXIT
B1A8 4300		353	CRFIX: ORL A,000H ;SET BIT 7
B1AA 03		354	RET ;EXIT
B1AB FD		355	LFFIX: MOV A,STATUS ;GET THE STATUS
B1AC 4320		356	ORL A,02BH ;SET LF BIT IN STATUS
B1AE AD		357	MOV STATUS,A ;PUT THE STATUS BACK
B1AF 2320		358	MOV A,02BH ;GET A SPACE
B1B1 03		359	RET ;EXIT
B1B2 FD		360	FFFIX: MOV A,STATUS ;GET THE STATUS
B1B3 4320		361	ORL A,02BH ;SET LINE FEED BIT
B1B5 AD		362	MOV STATUS,A ;PUT THE STATUS BACK
B1B6 FE		363	MOV A,LINCNT ;GET THE LINE COUNT
B1B7 4300		364	ORL A,000H ;SET BIT 7
B1B9 AE		365	MOV LINCNT,A ;PUT LINE COUNT BACK
B1BA 2320		366	MOV A,02BH ;GET A SPACE
B1BC 03		367	RET ;EXIT
B1BD FF		368	ISCHAR: MOV A,JUNK1 ;GET CHARACTER BACK
B1BE 533F		369	AHL A,03FH ;STRIP THE TWO MSB
B1C0 03		370	RET ;EXIT

LOC	OBJ	SEQ	SOURCE STATEMENT	INSTR	STATUS	LOC	OBJ
		371	;				
		372	;				
		373	;				
B1C1	AC	374	PRNTIT: MOV A, TEMP1	MOV			
B1C2	E7	375	RL YRA	RL			
B1C3	E7	376	RL ROT10	RL			
B1C4	6C	377	ADD A, TEMP1	ADD			
		378	;				
		379	;				
		380	;				
B1C5	2C	381	XCH A, TEMP1	XCH			
B1C6	B2CA	382	JB5 SHORT	JB			
B1C8	44AB	383	JMP PAGE1	JMP			
B1CA	64AB	384	SHORT: JMP PAGE2	JMP			
		385	;				
		386	;				
		387	;				
		388	;				
B1CC	AF	389	FIRE: MOV JUNK1, A	MOV			
B1CD	FD	390	MOV A, STATUS	MOV			
B1CE	D2D4	391	JB6 NT1	JB			
B1DB	56DB	392	FIREX: JT1 FIREX	JT			
B1DE	24D6	393	JMP FIREY	JMP			
B1D4	46D4	394	NT1: JNT1 NT1	JNT			
B1D6	FF	395	FIREY: MOV A, JUNK1	MOV			
B1D7	9B	396	MOVX PRB, A	MOVX			
		397	;				
		398	;				
		399	;				
B1D8	23F3	400	MOV A, #BF3H	MOV			
B1DA	62	401	MOV T, A	MOV			
B1DB	55	402	START T	START			
B1DC	16EB	403	TSJTF: JTF KTDUN	JTF			
B1DE	24DC	404	JMP TSJTF	JMP			
B1EB	27	405	KTDUN: CLR A	CLR			
B1E1	9B	406	MOVX PRB, A	MOVX			
B1E2	65	407	STOP TCNT	STOP			
B1E3	93	408	RET	RET			
		409	*EJECT				

1-190

LOC	OBJ	SEQ	SOURCE STATEMENT	TRANSLATED STATEMENT	LOC	OBJ
		457			510	
B21E 7F		458	DB *****7FH	: ***** 00	511	3C 8050
B21F 09		459	DB *****89H	: ***** 00	512	24 3050
B220 09		460	DB *****89H	: ***** 00	513	14 0050
B221 09		461	DB *****89H	: ***** 00	514	14 3050
B222 01		462	DB *****B1H	: ***** 00	515	0E 1050
		463			516	
B223 3E		464	DB *****3EH	: ***** 00	517	77 0050
B224 41		465	DB *****41H	: ***** 00	518	00 1050
B225 41		466	DB *****41H	: ***** 00	519	00 0050
B226 51		467	DB *****51H	: ***** 00	520	00 0050
B227 71		468	DB *****71H	: ***** 00	521	00 0050
		469			522	
B228 7F		470	DB *****7FH	: ***** 00	523	30 0050
B229 00		471	DB *****80H	: ***** 00	524	14 0050
B22A 00		472	DB *****80H	: ***** 00	525	10 0050
B22B 00		473	DB *****80H	: ***** 00	526	10 0050
B22C 7F		474	DB *****7FH	: ***** 00	527	32 0050
		475			528	
B22D 00		476	DB *****80H	: ***** 00	529	77 0050
B22E 41		477	DB *****41H	: ***** 00	530	70 0050
B22F 7F		478	DB *****7FH	: ***** 00	531	01 0050
B230 41		479	DB *****41H	: ***** 00	532	00 0050
B231 00		480	DB *****80H	: ***** 00	533	00 0050
		481			534	00 0050
B232 20		482	DB *****20H	: ***** 00	535	45 0050
B233 40		483	DB *****40H	: ***** 00	536	70 0050
B234 40		484	DB *****40H	: ***** 00	537	00 1050
B235 40		485	DB *****40H	: ***** 00	538	00 0050
B236 3F		486	DB *****3FH	: ***** 00	539	01 0050
		487			540	
B237 7F		488	DB *****7FH	: ***** 00	541	10 0050
B238 00		489	DB *****80H	: ***** 00	542	10 0050
B239 14		490	DB *****14H	: ***** 00	543	77 0050
B23A 22		491	DB *****22H	: ***** 00	544	10 0050
B23B 41		492	DB *****41H	: ***** 00	545	10 0050
		493			546	
B23C 7F		494	DB *****7FH	: ***** 00	547	70 0050
B23D 40		495	DB *****40H	: ***** 00	548	00 0050
B23E 40		496	DB *****40H	: ***** 00	549	00 0050
B23F 40		497	DB *****40H	: ***** 00	550	00 0050
B240 40		498	DB *****40H	: ***** 00	551	70 0050
		499			552	
B241 7F		500	DB *****7FH	: ***** 00	553	71 0050
B242 02		501	DB *****02H	: ***** 00	554	00 1050
B243 0C		502	DB *****0CH	: ***** 00	555	00 0050
B244 02		503	DB *****02H	: ***** 00	556	00 1050
B245 7F		504	DB *****7FH	: ***** 00	557	71 0050
		505			558	
B246 7F		506	DB *****7FH	: ***** 00	559	77 0050
B247 04		507	DB *****04H	: ***** 00	560	00 1050
B248 00		508	DB *****80H	: ***** 00	561	00 0050
B249 10		509	DB *****10H	: ***** 00	562	00 0050
B24A 7F		510	DB *****7FH	: ***** 00	563	77 0050
		511	#EJECT			

LOC	OBJ	SEQ	SOURCE STATEMENT	TABLET 33W00	072	180 00J
		512				
024B	3E	513	DB ---- 3EH	: H***** 00	02A	75 3100
024C	41	514	DB ---- 41H	: H* * * 00	02A	40 3100
024D	41	515	DB ---- 41H	: H* * * 00	02A	40 3100
024E	41	516	DB ---- 41H	: H* * * 00	13A	40 3100
024F	3E	517	DB ---- 3EH	: H***** 00	53A	10 3100
		518				
025B	7F	519	DB ---- 7FH	: H***** 00	03A	30 3100
0251	09	520	DB ---- 09H	: H10 * * 00	03A	10 3100
0252	09	521	DB ---- 09H	: H10 * * 00	03A	10 3100
0253	09	522	DB ---- 09H	: H10 * * 00	13A	10 3100
0254	06	523	DB ---- 06H	: H15 * * 00	03A	15 3100
		524				
0255	3E	525	DB ---- 3EH	: H***** 00	03A	75 3100
0256	41	526	DB ---- 41H	: H* * * 00	13A	40 3100
0257	51	527	DB ---- 51H	: H* * * 00	03A	40 3100
0258	21	528	DB ---- 21H	: H* * * 00	03A	40 3100
0259	5E	529	DB ---- 5EH	: H* ***** 00	03A	75 3100
		530				
025A	7F	531	DB ---- 7FH	: H***** 00	03A	40 3100
025B	09	532	DB ---- 09H	: H10 * * 00	13A	10 3100
025C	19	533	DB ---- 19H	: H10 * * 00	03A	75 3100
025D	29	534	DB ---- 29H	: H10 * * 00	03A	10 3100
025E	46	535	DB ---- 46H	: H15 * * 00	03A	40 3100
		536				
025F	26	537	DB ---- 26H	: H* * * 00	13A	40 3100
0260	49	538	DB ---- 49H	: H* * * 00	03A	40 3100
0261	49	539	DB ---- 49H	: H* * * 00	03A	40 3100
0262	49	540	DB ---- 49H	: H* * * 00	03A	40 3100
0263	32	541	DB ---- 32H	: H10 * * 00	03A	75 3100
		542				
0264	01	543	DB ---- 01H	: H01 * * 00	03A	75 3100
0265	01	544	DB ---- 01H	: H01 * * 00	03A	40 3100
0266	7F	545	DB ---- 7FH	: H***** 00	03A	40 3100
0267	01	546	DB ---- 01H	: H01 * * 00	13A	40 3100
0268	01	547	DB ---- 01H	: H01 * * 00	03A	10 3100
		548				
0269	3F	549	DB ---- 3FH	: H***** 00	03A	75 3100
026A	4B	550	DB ---- 4BH	: H* * * 00	03A	40 3100
026B	4B	551	DB ---- 4BH	: H* * * 00	03A	40 3100
026C	4B	552	DB ---- 4BH	: H* * * 00	13A	40 3100
026D	3F	553	DB ---- 3FH	: H***** 00	03A	40 3100
		554				
026E	1F	555	DB ---- 1FH	: H01 ***** 00	030	75 3100
026F	20	556	DB ---- 20H	: H02 * * 00	100	30 3100
0270	40	557	DB ---- 40H	: H02 * * 00	030	30 3100
0271	20	558	DB ---- 20H	: H02 * * 00	030	30 3100
0272	1F	559	DB ---- 1FH	: H01 ***** 00	030	75 3100
		560				
0273	7F	561	DB ---- 7FH	: H***** 00	030	75 3100
0274	20	562	DB ---- 20H	: H* * * 00	100	40 3100
0275	10	563	DB ---- 10H	: H00 * * 00	030	40 3100
0276	20	564	DB ---- 20H	: H* * * 00	030	40 3100
0277	7F	565	DB ---- 7FH	: H***** 00	030	75 3100
		566	\$EJECT			

LOC	OBJ	SEQ	SOURCE STATEMENT	TRANSLATE	33000	032	600	20J
		567						
0270	63	568	DBRT 663H	DBRT 663H	DBRT 663H	DBRT 663H	DBRT 663H	DBRT 663H
0271	14	569	DBRT 14H	DBRT 14H	DBRT 14H	DBRT 14H	DBRT 14H	DBRT 14H
027A	08	570	DBRT 88H	DBRT 88H	DBRT 88H	DBRT 88H	DBRT 88H	DBRT 88H
027B	14	571	DBRT 14H	DBRT 14H	DBRT 14H	DBRT 14H	DBRT 14H	DBRT 14H
027C	63	572	DBRT 63H	DBRT 63H	DBRT 63H	DBRT 63H	DBRT 63H	DBRT 63H
		573						
027D	03	574	DBRT 03H	DBRT 03H	DBRT 03H	DBRT 03H	DBRT 03H	DBRT 03H
027E	04	575	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H
027F	70	576	DBRT 70H	DBRT 70H	DBRT 70H	DBRT 70H	DBRT 70H	DBRT 70H
0280	04	577	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H
0281	03	578	DBRT 03H	DBRT 03H	DBRT 03H	DBRT 03H	DBRT 03H	DBRT 03H
		579						
0282	61	580	DBRT 61H	DBRT 61H	DBRT 61H	DBRT 61H	DBRT 61H	DBRT 61H
0283	51	581	DBRT 51H	DBRT 51H	DBRT 51H	DBRT 51H	DBRT 51H	DBRT 51H
0284	49	582	DBRT 49H	DBRT 49H	DBRT 49H	DBRT 49H	DBRT 49H	DBRT 49H
0285	45	583	DBRT 45H	DBRT 45H	DBRT 45H	DBRT 45H	DBRT 45H	DBRT 45H
0286	43	584	DBRT 43H	DBRT 43H	DBRT 43H	DBRT 43H	DBRT 43H	DBRT 43H
		585						
0287	7F	586	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH
0288	7F	587	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH
0289	41	588	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H
028A	41	589	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H
028B	41	590	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H
		591						
028C	02	592	DBRT 02H	DBRT 02H	DBRT 02H	DBRT 02H	DBRT 02H	DBRT 02H
028D	04	593	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H
028E	08	594	DBRT 08H	DBRT 08H	DBRT 08H	DBRT 08H	DBRT 08H	DBRT 08H
028F	10	595	DBRT 10H	DBRT 10H	DBRT 10H	DBRT 10H	DBRT 10H	DBRT 10H
0290	20	596	DBRT 20H	DBRT 20H	DBRT 20H	DBRT 20H	DBRT 20H	DBRT 20H
		597						
0291	41	598	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H
0292	41	599	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H
0293	41	600	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H	DBRT 41H
0294	7F	601	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH
0295	7F	602	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH	DBRT 7FH
		603						
0296	10	604	DBRT 10H	DBRT 10H	DBRT 10H	DBRT 10H	DBRT 10H	DBRT 10H
0297	08	605	DBRT 08H	DBRT 08H	DBRT 08H	DBRT 08H	DBRT 08H	DBRT 08H
0298	04	606	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H	DBRT 04H
0299	08	607	DBRT 08H	DBRT 08H	DBRT 08H	DBRT 08H	DBRT 08H	DBRT 08H
029A	10	608	DBRT 10H	DBRT 10H	DBRT 10H	DBRT 10H	DBRT 10H	DBRT 10H
		609						
029B	40	610	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H
029C	40	611	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H
029D	40	612	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H
029E	40	613	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H
029F	40	614	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H	DBRT 40H
		615	*EJECT					

1

02A0 8888	616	:	STBCNT, 088H	:ZERO STROBE COUNTER	000	00 8750
02A2 FA	617 PAGE1:	MOV	A, SAVPNT	:GET DIRECTION	000	01 8750
02A3 37	618	MOV	A, SAVPNT	:FLIP BITS	000	00 8750
02A4 D2B3	619	CPL	A	:IF BACKWARD JUMP, OUT	000	01 8750
02A6 FC	620	JB6	BAKWRD	:GET THE TARGET	000	00 8750
02A7 A3	621 LKLO:	MOV	A, TEMP1	:GET THE DATA	000	00 8750
02A8 34CC	622	MOV	A, 0A	:STROBE THE SOLENOIDS	000	00 8750
02AA 1C	623	CALL	FIRE	:INCREMENT THE POINTER	000	01 8750
02AB 18	624	INC	TEMP1	:INCREMENT THE STROBE COUNTER	000	01 8750
02AC FB	625	INC	STBCNT	:GET THE STROBE COUNTER	000	00 8750
02AD D305	626	MOV	A, STBCNT	:IS IT FIVE	000	00 8750
02AF 96A6	627	XRL	A, 0B5H	:REPEAT IF NOT FIVE	000	00 8750
02B1 04AE	628	JNZ	LKLO	:GO BACK	000	00 8750
02B3 FC	629	JMP	SETTIM	:GET THE TARGET	000	00 8750
02B4 B3B4	630 BAKWRD:	MOV	A, TEMP1	:COMPENSATE FOR GOING BACKWARDS	000	00 8750
02B6 AC	631	ADD	A, 0B4H	:SAVE IT	000	00 8750
02B7 FC	632	MOV	TEMP1, A	:GET THE TARGET	000	00 8750
02B8 A3	633 LKLO1:	MOV	A, TEMP1	:GET THE DATA	000	00 8750
02B9 34CC	634	MOV	A, 0A	:STROBE THE SOLENOIDS	000	00 8750
02BB FC	635	CALL	FIRE	:GET TEMP1	000	00 8750
02BC 07	636	MOV	A, TEMP1	:DECREASE BY ONE	000	00 8750
02BD AC	637	DEC	A	:PUT IT BACK	000	00 8750
02BE 18	638	MOV	TEMP1, A	:INCREMENT THE STROBE COUNTER	000	00 8750
02BF FB	639	INC	STBCNT	:GET THE STROBE COUNTER	000	00 8750
02C0 D305	640	MOV	A, STBCNT	:IS IT FIVE	000	00 8750
02C2 96B7	641	XRL	A, 0B5H	:REPEAT IF NOT FIVE	000	00 8750
02C4 04AE	642	JNZ	LKLO1	:GO BACK, CHARACTER IS DONE	000	00 8750
	643	JMP	SETTIM			
	644 \$EJECT					

1

LDC	OBJ	SEQ	SOURCE STATEMENT	INSTRUMENT CIRCUIT	Q82	LDC	OBJ
		691					
0323	00	692	DB 00H	0000 00	000		0000
0324	00	693	DB 00H	0000 00	000		0000
0325	07	694	DB 07H	0000 00	000		0000
0326	00	695	DB 00H	0000 00	000	00	0000
0327	00	696	DB 00H	0000 00	000	00	0000
		697					
0328	1C	698	DB 1CH	0000 00	000	00	0000
0329	22	699	DB 22H	0000 00	000	00	0000
032A	41	700	DB 41H	0000 00	000	00	0000
032B	00	701	DB 00H	0000 00	000	00	0000
032C	00	702	DB 00H	0000 00	000	00	0000
		703					
032D	00	704	DB 00H	0000 00	000	00	0000
032E	00	705	DB 00H	0000 00	000	00	0000
032F	41	706	DB 41H	0000 00	000	00	0000
0330	22	707	DB 22H	0000 00	000	00	0000
0331	1C	708	DB 1CH	0000 00	000	00	0000
		709					
0332	22	710	DB 22H	0000 00	000	00	0000
0333	14	711	DB 14H	0000 00	000	00	0000
0334	7F	712	DB 7FH	0000 00	000	00	0000
0335	14	713	DB 14H	0000 00	000	00	0000
0336	22	714	DB 22H	0000 00	000	00	0000
		715					
0337	00	716	DB 00H	0000 00	000	00	0000
0338	00	717	DB 00H	0000 00	000	00	0000
0339	7F	718	DB 7FH	0000 00	000	00	0000
033A	00	719	DB 00H	0000 00	000	00	0000
033B	00	720	DB 00H	0000 00	000	00	0000
		721					
033C	00	722	DB 00H	0000 00	000	00	0000
033D	40	723	DB 40H	0000 00	000	00	0000
033E	30	724	DB 30H	0000 00	000	00	0000
033F	00	725	DB 00H	0000 00	000	00	0000
0340	00	726	DB 00H	0000 00	000	00	0000
		727					
0341	00	728	DB 00H	0000 00	000	00	0000
0342	00	729	DB 00H	0000 00	000	00	0000
0343	00	730	DB 00H	0000 00	000	00	0000
0344	00	731	DB 00H	0000 00	000	00	0000
0345	00	732	DB 00H	0000 00	000	00	0000
		733					
0346	00	734	DB 00H	0000 00	000	00	0000
0347	00	735	DB 00H	0000 00	000	00	0000
0348	40	736	DB 40H	0000 00	000	00	0000
0349	00	737	DB 00H	0000 00	000	00	0000
034A	00	738	DB 00H	0000 00	000	00	0000
		739					
034B	20	740	DB 20H	0000 00	000	00	0000
034C	10	741	DB 10H	0000 00	000	00	0000
034D	00	742	DB 00H	0000 00	000	00	0000
034E	04	743	DB 04H	0000 00	000	00	0000
034F	02	744	DB 02H	0000 00	000	00	0000
		745					
0350	3E	746	DB 3EH	0000 00	000	00	0000
0351	51	747	DB 51H	0000 00	000	00	0000
0352	49	748	DB 49H	0000 00	000	00	0000
0353	45	749	DB 45H	0000 00	000	00	0000
0354	3E	750	DB 3EH	0000 00	000	00	0000
		751					
0355	00	752	DB 00H	0000 00	000	00	0000
0356	42	753	DB 42H	0000 00	000	00	0000
0357	7F	754	DB 7FH	0000 00	000	00	0000
0358	40	755	DB 40H	0000 00	000	00	0000
0359	00	756	DB 00H	0000 00	000	00	0000
		757					
035A	62	758	DB 62H	0000 00	000	00	0000
035B	51	759	DB 51H	0000 00	000	00	0000
035C	49	760	DB 49H	0000 00	000	00	0000
035D	49	761	DB 49H	0000 00	000	00	0000
035E	46	762	DB 46H	0000 00	000	00	0000
		763					
035F	21	764	DB 21H	0000 00	000	00	0000
0360	41	765	DB 41H	0000 00	000	00	0000

1

037D 46	000	DB	46H	:	00	00%	00 1000
037E 49	001	DB	49H	:	00	00%	00 0000
037F 49	002	DB	49H	:	00	00%	00 0000
0380 29	003	DB	29H	:	00	00%	00 0000
0381 1E	004	DB	1EH	:	00	00%	00 0000
	005	DB	00H	:	00	00%	00 0000
	006	DB	00H	:	00	00%	00 0000
0382 00	007	DB	00H	:	00	00%	00 0000
0383 00	008	DB	00H	:	00	00%	00 0000
0384 14	009	DB	14H	:	00	00%	00 0000
0385 00	010	DB	00H	:	00	00%	00 0000
0386 00	011	DB	00H	:	00	00%	00 0000
	012	DB	00H	:	00	00%	00 0000
0387 00	013	DB	00H	:	00	00%	00 0000
0388 40	014	DB	40H	:	00	00%	00 0000
0389 34	015	DB	34H	:	00	00%	00 0000
038A 00	016	DB	00H	:	00	00%	00 0000
038B 00	017	DB	00H	:	00	00%	00 0000
	018	DB	00H	:	00	00%	00 0000
038C 00	019	DB	00H	:	00	00%	00 0000
038D 14	020	DB	14H	:	00	00%	00 0000
038E 22	021	DB	22H	:	00	00%	00 0000
038F 41	022	DB	41H	:	00	00%	00 0000
0390 00	023	DB	00H	:	00	00%	00 0000
	024	DB	14H	:	00	00%	00 0000
0391 14	025	DB	14H	:	00	00%	00 0000
0392 14	026	DB	14H	:	00	00%	00 0000
0393 14	027	DB	14H	:	00	00%	00 0000
0394 14	028	DB	14H	:	00	00%	00 0000
0395 14	029	DB	14H	:	00	00%	00 0000
	030	DB	00H	:	00	00%	00 0000
0396 00	031	DB	00H	:	00	00%	00 0000
0397 41	032	DB	41H	:	00	00%	00 0000
0398 22	033	DB	22H	:	00	00%	00 0000
0399 14	034	DB	14H	:	00	00%	00 0000
039A 00	035	DB	00H	:	00	00%	00 0000
	036	DB	02H	:	00	00%	00 0000
039B 02	037	DB	02H	:	00	00%	00 0000
039C 01	038	DB	01H	:	00	00%	00 0000
039D 59	039	DB	59H	:	00	00%	00 0000
039E 05	040	DB	05H	:	00	00%	00 0000
039F 02	041	DB	02H	:	00	00%	00 0000
	042	*EJECT					

LDC	OBJ	SEQ	SOURCE STATEMENT	THIRDTATE 319002	930	(60 30)
03A8	B8B8	843	PAGE2: MOV STBCNT, #00H	:ZERO STROBE COUNTER	000	
03A2	FA	844	MOV A, SAVPNT	:GET DIRECTION	000	0000
03A3	37	845	CPL A	:FLIP BITS	000	
03A4	D2B5	846	JBG 37A BKWRD	:IF BACKWARD JUMP OUT	000	TS 0000
03A6	FC	847	LKHI:02 MOV A, TEMP1	:GET THE TARGET	000	00 0000
03A7	B36B	848	ADD A, #6BH	:ADJUST THE TARGET	000	0000 0000
03A9	A3	849	MOV A, #A	:GET THE DATA	000	0000 0000
03AA	34CC	850	CALL FIRE	:STROBE THE SOLENOIDS	000	0000 0000
03AC	1C	851	INC TEMP1	:INCREMENT THE POINTER	000	
03AD	1B	852	INC STBCNT	:INCREMENT THE STROBE COUNTER	000	0000 0000
03AE	FB	853	MOV A, STBCNT	:GET THE STROBE COUNTER	000	00 0000
03AF	D3B5	854	XRL A, #05H	:IS IT FIVE	000	
03B1	96A6	855	JNZ LKHI	:REPEAT IF NOT FIVE	000	
03B3	84AE	856	JMP SETTIM	:GO BACK	000	
03B5	FC	857	BKWRD: MOV A, TEMP1	:GET THE TARGET	000	0000 0000
03B6	B364	858	ADD A, #64H	:COMPENSATE FOR GOING BACKWARDS	000	0000 0000
03B8	AC	859	MOV A, TEMP1	:SAVE IT	000	0000 0000
03B9	FC	860	LKHI: MOV A, TEMP1	:GET THE TARGET	000	00 0000
03BA	A3	861	MOV A, #A	:GET THE DATA	000	00 0000
03BB	34CC	862	CALL FIRE	:STROBE THE SOLENOIDS	000	0000 0000
03BD	FC	863	MOV A, TEMP1	:GET TEMP1	000	0000 0000
03BE	B7	864	DEC A	:DECREASE BY ONE	000	0000 0000
03BF	AC	865	MOV A, TEMP1	:PUT IT BACK	000	0000 0000
03C0	1B	866	INC STBCNT	:INCREMENT THE STROBE COUNTER	000	0000 0000
03C1	FB	867	MOV A, STBCNT	:GET THE STROBE COUNTER	000	
03C2	D3B5	868	XRL A, #05H	:IS IT FIVE	000	
03C4	96B9	869	JNZ LKHI	:REPEAT IF NOT FIVE	000	
03C6	84AE	870	JMP SETTIM	:GO BACK, CHARACTER IS DONE	000	0000 0000
		871	*EJECT		000	0000 0000

LOC	OBJ	SEQ	SOURCE STATEMENT	TARGET STATE	FROM	TO
		872	RETURN: MOV A, #0010	8000	10000	0000
0400		873	ORG 400H	0000	10000	0000
		874		0000	10000	0000
0400 27		875	BGIN: CLR A	0000	10000	0000
0401 98		876	MOVX PRB, A	0000	10000	0000
0402 9400		877	CALL SETUP	0000	10000	0000
0404 943F		878	CALL VARSET	0000	10000	0000
0406 040A		879	JMP PRMT	0000	10000	0000
		880		0000	10000	0000
0408 23FE		881	SETUP: MOV A, #0FEH	0000	10000	0000
040A 39		882	OUTL A, P1	0000	10000	0000
		883		0000	10000	0000
		884	MOV DELAY, 3.2 SECONDS	0000	10000	0000
		885		0000	10000	0000
0408 BC85		886	MOV TEMP1, #05H	0000	10000	0000
040D BFFF		887	SELF: MOV JUNK1, #0FFH	0000	10000	0000
040F BEFF		888	SELF: MOV LINCNT, #0FFH	0000	10000	0000
0411 09		889	SELF: IN A, P1	0000	10000	0000
0412 37		890	CPL A	0000	10000	0000
0413 F21D		891	JB7 DONE1	0000	10000	0000
0415 EE11		892	DJNZ LINCNT, SELF	0000	10000	0000
0417 EBF8		893	DJNZ JUNK1, SELF	0000	10000	0000
0419 ECBD		894	DJNZ TEMP1, SELF	0000	10000	0000
041B 045A		895	JMP ERROR	0000	10000	0000
		896		0000	10000	0000
		897	MOV MAKE_SURE, 7	0000	10000	0000
		898		0000	10000	0000
041D BFFF		899	DONE1: MOV JUNK1, #0FFH	0000	10000	0000
041F BEFF		900	SELF: MOV LINCNT, #0FFH	0000	10000	0000
0421 09		901	SELF: IN A, P1	0000	10000	0000
0422 F22A		902	JB7 DONE1	0000	10000	0000
0424 EE21		903	DJNZ LINCNT, SELF	0000	10000	0000
0426 EF1F		904	DJNZ JUNK1, SELF	0000	10000	0000
0428 045A		905	JMP ERROR	0000	10000	0000
		906		0000	10000	0000
		907	MOV CHECK_LEFT, 7	0000	10000	0000
		908		0000	10000	0000
042A BC84		909	DONE1: MOV TEMP1, #04H	0000	10000	0000
042C BFFF		910	SELF: MOV JUNK1, #0FFH	0000	10000	0000
042E BEFF		911	SELF: MOV LINCNT, #0FFH	0000	10000	0000
0430 09		912	SELF: IN A, P1	0000	10000	0000
0431 37		913	CPL A	0000	10000	0000
0432 D23C		914	JB6 DONE1	0000	10000	0000
0434 EE3B		915	DJNZ LINCNT, SELF	0000	10000	0000
0436 EF2E		916	DJNZ JUNK1, SELF	0000	10000	0000
0438 EC2C		917	DJNZ TEMP1, SELF	0000	10000	0000
043A 045A		918	JMP ERROR	0000	10000	0000
043C 0901		919	DONE1: ORL P1, #01H	0000	10000	0000
043E 03		920	RET	0000	10000	0000
		921		0000	10000	0000
		922		0000	10000	0000
		923	MOV SET_UP_VARIABLES, 7	0000	10000	0000
		924		0000	10000	0000
043F 23FE		925	VARSET: MOV A, #0FEH	0000	10000	0000
0441 62		926	MOV T, A	0000	10000	0000
0442 55		927	STRT T	0000	10000	0000
0443 082B		928	MOV INBUF, #FIRST	0000	10000	0000
0445 08BB		929	MOV LINCNT, #0BH	0000	10000	0000
0447 08BB		930	MOV STATUS, #0BH	0000	10000	0000
		931		0000	10000	0000
		932	MOV CLEAR_RAM, 7	0000	10000	0000
		933		0000	10000	0000
0449 092B		934	MOV OUTBUF, #FIRST	0000	10000	0000
044B 232B		935	CLRMEM: MOV A, #2BH	0000	10000	0000
044D A1		936	MOV OUTBUF, A	0000	10000	0000
044E 19		937	INC OUTBUF	0000	10000	0000
044F F9		938	MOV A, OUTBUF	0000	10000	0000
0450 D37B		939	XRL A, #MAX+1	0000	10000	0000
0452 364B		940	JNZ CLRMEM	0000	10000	0000
		941		0000	10000	0000
		942	MOV CLEAR_8212, 7	0000	10000	0000
		943		0000	10000	0000
0454 99EF		944	ANL P1, #0FEH	0000	10000	0000
0456 0B		945	MOVX A, @INBUF	0000	10000	0000
0457 091B		946	ORL P1, #0BH	0000	10000	0000
		947		0000	10000	0000

LDC OBJ	SEQ	SOURCE STATEMENT	TRANSISTE	STATUS	LOC	OBJ
	948	:NOW EXIT VARSET				
	949	: ;				
B459 83	950	RET				
	951	: ;				
	952	:THIS ROUTINE TURNS THE MOTOR OFF AND LOOPS				
	953	: ;				
B45A 89FF	954	ERROR: ORL P1,#0FFH				
B45C 845C	955	DEAD: JMP DEAD				
	956	: ;				
	957	:THESE ARE ALL SUBROUTINES THAT ARE CALLED				
	958	: ;				
B45E 19	959	INCTST: INC OUTBUF				
B45F 237B	960	MOV A,#MAX-1				
B461 09	961	XRL A,OUTBUF				
B462 83	962	RET				
B463 89	963	GTPRNT: IN A,P1				
B464 37	964	CPL A				
B465 0263	965	JB6 GTPRNT				
B467 166B	966	TSTJTF: JTF PIT				
B469 8467	967	JMP TSTJTF				
B46B 65	968	PIT: STOP TCNT				
B46C FF	969	MOV A,JUNK1				
B46D 34C1	970	CALL PRNTIT				
B46F 341C	971	CALL JYLNMOE				
B471 83	972	RET				
B472 F9	973	DECTST: MOV OUTBUF				
B473 87	974	DEC INTA				
B474 A9	975	MOV OUTBUF,A				
B475 D31F	976	XRL A,#FIRST-1				
B477 83	977	RET				
	978	: ;				
	979	:THIS ROUTINE DOES A LINE FEED				
	980	: ;				
B47B FE	981	LINEFD: MOV A,LINCNT				
B479 F298	982	JB7 DOFF				
B47B 99F0	983	LFDO: ANL P1,#BFDH				
B47D BC40	984	MOV TEMP1,#4DH				
B47F BF93	985	LFLP1: MOV JUNK1,#93H				
B481 EF81	986	LFLP2: DJNZ JUNK1,LFLP2				
B483 EC7F	987	DJNZ TEMP1,LFLP1				
B485 89B2	988	ORL A,P1,#B2H				
B487 1E	989	INC LINCNT				
B488 FE	990	MOV A,LINCNT				
B489 D328	991	XRL A,#28H				
B48B 968F	992	JNZ NOTDON				
B48D BE8B	993	MOV A,LINCNT #8BH				
	994	: ;				
	995	:NOW DELAY 98 MILLISECDS				
	996	: ;				
B48F BC8B	997	NOTDON: MOV TEMP1,#8BH				
B491 BFFF	998	LDP1: MOV JUNK1,#BFFH				
B493 EF93	999	LDP2: DJNZ JUNK1,LDP2				
B495 EC91	1000	DJNZ TEMP1,LDP1				
B497 83	1001	RET				
	1002	: ;				
	1003	:THIS ROUTINE DOES A FORM FEED				
	1004	: ;				
B498 89	1005	DOFF: IN A,P1				
B499 37	1006	CPL A				
B49A 53C8	1007	ANL A,#BCBH				
B49C C698	1008	JZ DOFF				
B49E 8981	1009	ORL P1,#81H				
B4A0 9476	1010	CALL LFDO				
B4A2 FE	1011	FFCK: MOV A,LINCNT				
B4A3 537F	1012	ANL A,#7FH				
B4A5 D38B	1013	XRL A,#8BH				
B4A7 C6A0	1014	JZ FFDONE				
B4A9 9478	1015	CALL LFDO				
B4AB 84A2	1016	JMP FFCY				
B4AD 83	1017	FFDONE: RET				
	1018	: ;				
B4AE 23EB	1019	SETTIM: MOV A,#8EBH				
B4B0 62	1020	MOV T,A				
B4B1 55	1021	STRT T				
B4B2 83	1022	RET				
	1023	: ;				

1-202

Article Reprints

Article Reprints
MCS®-51 Application Notes & 2

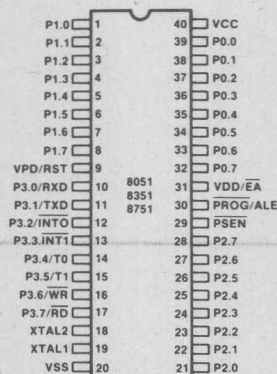


Figure 1a. 8051 Microcomputer Pinout Diagram

1. INTRODUCTION

In 1976 Intel introduced the MCS-48™ family, consisting of the 8048, 8748, and 8035 microcomputers. These parts marked the first time a complete microcomputer system, including an eight-bit CPU, 1024 8-bit words of ROM or EPROM program memory, 64 words of data memory, I/O ports and an eight-bit timer counter could be integrated onto a single silicon chip. Depending only on the program memory contents, one chip could control a limitless variety of products, ranging from appliances or automobile engines to text or data processing equipment. Follow-on products stretched the MCS-48™ architecture in several directions: the 8049 and 8039 doubled the amount of on-chip memory and ran 83% faster; the 8021 reduced costs by executing a subset of the 8048 instructions with a somewhat slower clock; and the 8022 put a unique two-channel 8-bit analog-to-digital converter on the same NMOS chip as the computer, letting the chip interface directly with analog transducers.

Now three new high-performance single-chip microcomputers—the Intel® 8051, 8751, and 8031—extend the advantages of Integrated Electronics to whole new product areas. Thanks to Intel's new HMOS technology, the MCS-51™ family provides four times the program memory and twice the data memory as the 8048 on a single chip. New I/O and peripheral capabilities both increase the range of applicability and reduce total system cost. Depending on the use, processing throughput increases by two and one-half to ten times.

This Application Note is intended to introduce the reader to the MCS-51™ architecture and features. While it does not assume intimacy with the MCS-48™ product line on the part of the reader, he/she should be familiar with

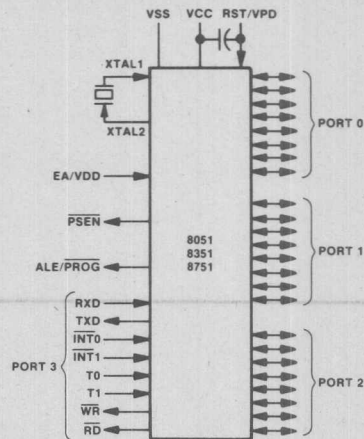


Figure 1b. 8051 Microcomputer Logic Symbol

some microprocessor (preferably Intel's, of course) or have a background in computer programming and digital logic.

Family Overview

Pinout diagrams for the 8051, 8751, and 8031 are shown in Figure 1. The devices include the following features:

- Single-supply 5 volt operation using HMOS technology.
- 4096 bytes program memory on-chip (not on 8031).
- 128 bytes data memory on-chip.
- Four register banks.
- 128 User-defined software flags.
- 64 Kilobytes each program and external RAM addressability.
- One microsecond instruction cycle with 12 MHz crystal.
- 32 bidirectional I/O lines organized as four 8-bit ports (16 lines on 8031).
- Multiple mode, high-speed programmable Serial Port.
- Two multiple mode, 16-bit Timer/Counters.
- Two-level prioritized interrupt structure.
- Full depth stack for subroutine return linkage and data storage.
- Augmented MCS-48™ instruction set.
- Direct Byte and Bit addressability.
- Binary or Decimal arithmetic.
- Signed-overflow detection and parity computation.
- Hardware Multiple and Divide in 4 μ sec.
- Integrated Boolean Processor for control applications.
- Upwardly compatible with existing 8048 software.

Intel devices come in a standard 40-pin Dual In-Line Package, with the same pin-out, the same timing, and the same electrical characteristics. The primary difference between the three is the on-chip program memory—different types are offered to satisfy differing user requirements.

The 8751 provides 4K bytes of ultraviolet-Erasable, Programmable Read Only Memory (EPROM) for program development, prototyping, and limited production runs. (By convention, 1K means $2^{10} = 1024$. 1k—with a lower case “k”—equals $10^3 = 1000$.) This part may be individually programmed for a specific application using Intel’s Universal PROM Programmer (UPP). If software bugs are detected or design specifications change the same part may be “erased” in a matter of minutes by exposure to ultraviolet light and reprogrammed with the modified code. This cycle may be repeated indefinitely during the design and development phase.

The final version of the software must be programmed into a large number of production parts. The 8051 has 4K bytes of ROM which are mask-programmed with the customer’s order when the chip is built. This part is considerably less expensive, but cannot be erased or altered after fabrication.

The 8031 does not have any program memory on-chip, but may be used with up to 64K bytes of external standard or multiplexed ROMs, PROMs, or EPROMs. The 8031 fits well in applications requiring significantly larger or smaller amounts of memory than the 4K bytes provided by its two siblings.

(The 8051 and 8751 automatically access external program memory for all addresses greater than the 4096 bytes on-chip. The External Access input is an override for all internal program memory—the 8051 and 8751 will each emulate an 8031 when pin 31 is low.)

Throughout this Note, “8051” is used as a generic term. Unless specifically stated otherwise, the point applies equally to all three components. Table 1 summarizes the quantitative differences between the members of the MCS-48™ and MCS-51™ families.

The remainder of this Note discusses the various MCS-51™ features and how they can be used. Software and/or hard-

ware application examples illustrate many of the concepts. Several isolated tasks (rather than one complete system design example) are presented in the hope that some of them will apply to the reader’s experiences or needs.

A document this short cannot detail all of a computer system’s capabilities. By no means will all the 8051 instructions be demonstrated; the intent is to stress new or unique MCS-51™ operations and instructions generally used in conjunction with each other. For additional hardware information refer to the Intel MCS-51™ Family User’s Manual, publication number 121517. The assembly language and use of ASM51, the MCS-51™ assembler, are further described in the MCS-51™ Macro Assembler User’s Guide, publication number 9800937.

The next section reviews some of the basic concepts of microcomputer design and use. Readers familiar with the 8048 may wish to skim through this section or skip directly to the next, “ARCHITECTURE AND ORGANIZATION.”

Microcomputer Background Concepts

Most digital computers use the binary (base 2) number system internally. All variables, constants, alphanumeric characters, program statements, etc., are represented by groups of binary digits (“bits”), each of which has the value 0 or 1. Computers are classified by how many bits they can move or process at a time.

The MCS-51™ microcomputers contain an eight-bit central processing unit (CPU). Most operations process variables eight bits wide. All internal RAM and ROM, and virtually all other registers are also eight bits wide. An eight-bit (“byte”) variable (shown in Figure 2) may assume one of $2^8 = 256$ distinct values, which usually represent integers between 0 and 255. Other types of numbers, instructions, and so forth are represented by one or more bytes using certain conventions.

For example, to represent positive and negative values, the most significant bit (D7) indicates the sign of the other seven bits—0 if positive, 1 if negative—allowing integer variables between -128 and +127. For integers with extremely large magnitudes, several bytes are manipulated together as “multiple precision” signed or unsigned integers—16, 24, or more bits wide.

Table 1. Features of Intel’s Single-Chip Microcomputers

EPROM Program Memory	ROM Program Memory	External Program Memory	Program Memory (Int/Max)	Data Memory (Bytes)	Instr. Cycle Time	Input/Output Pins	Interrupt Sources	Reg. Banks
	8021		1K 1K	64	8.4 μ Sec	21	0	1
	8022		2K 2K	64	8.4 μ Sec	28	2	1
8748	8048	8035	1K 4K	64	2.5 μ Sec	27	2	2
	8049	8039	2K 4K	128	1.36 μ Sec	27	2	2
8751	8051	8031	4K 64K	128	1.0 μ Sec	32	5	4

AFN-01502A-05

The letters "MCS" have traditionally indicated a system or family of compatible Intel® microcomputer components, including CPUs, memories, clock generators, I/O expanders, and so forth. The numerical suffix indicates the microprocessor or microcomputer which serves as the cornerstone of the family. Microcomputers in the MCS-48™ family currently include the 8048-series (8035, 8048, & 8748), the 8049-series (8039 & 8049), and the 8021 and 8022; the family also includes the 8243, an I/O expander compatible with each of the microcomputers. Each computer's CPU is derived from the 8048, with essentially the same architecture, addressing modes, and instruction set, and a single assembler (ASM48) serves each.

The first members of the MCS-51™ family are the 8051, 8751, and 8031. The architecture of the 8051-series, while derived from the 8048, is not strictly compatible; there are more addressing modes, more instructions, larger address spaces, and a few other hardware differences. In this Application Note the letters "MCS-51" are used when referring to architectural features of the 8051-series—features which would be included on possible future microcomputers based on the 8051 CPU. Such products could have different amounts of memory (as in the 8048/8049) or different peripheral functions (as in the 8021 and 8022) while leaving the CPU and instruction set intact. ASM51 is the assembler used by all microcomputers in the 8051 family.

Two digit decimal numbers may be "packed" in an eight-bit value, using four bits for the binary code of each digit. This is called Binary-Coded Decimal (BCD) representation, and is often used internally in programs which interact heavily with human beings.

Alphanumeric characters (letters, numbers, punctuation marks, etc.) are often represented using the American Standard Code for Information Interchange (ASCII) convention. Each character is associated with a unique seven-bit binary number. Thus one byte may represent

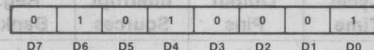


Figure 2. Representation of Bits Within an Eight-Bit "Byte" (Value shown = 01010001 Binary = 81 decimal).

a single character, and a word or sequence of letters may be represented by a series (or "string") of bytes. Since the ASCII code only uses 128 characters, the most significant bit of the byte is not needed to distinguish between characters. Often D7 is set to 0 for all characters. In some coding schemes, D7 is used to indicate the "parity" of the other seven bits—set or cleared as necessary to ensure that the total number of "1" bits in the eight-bit code is even ("even parity") or odd ("odd parity"). The 8051 includes hardware to compute parity when it is needed.

A computer program consists of an ordered sequence of specific, simple steps to be executed by the CPU one-at-a-time. The method or sequence of steps used collectively to solve the user's application is called an "algorithm."

The program is stored inside the computer as a sequence of binary numbers, where each number corresponds to one of the basic operations ("opcodes") which the CPU is capable of executing. In the 8051, each program memory location is one byte. A complete instruction consists of a sequence of one or more bytes, where the first defines the operation to be executed and additional bytes (if needed) hold additional information, such as data values or variable addresses. No instruction is longer than three bytes.

The way in which binary opcodes and modifier bytes are assigned to the CPU's operations is called the computer's "machine language." Writing a program directly in machine language is time-consuming and tedious. Human beings think in words and concepts rather than encoded numbers, so each CPU operation and resource is given a name and standard abbreviation ("mnemonic"). Programs are more easily discussed using these standard mnemonics, or "assembly language," and may be typed into an Intel® Intellec® 800 or Series II® microcomputer development system in this form. The development system can mechanically translate the program from assembly language "source" form to machine language "object" code using a program called an "assembler." The MCS-51™ assembler is called ASM51.

There are several important differences between a computer's machine language and the assembly language used as a tool to represent it. The machine language or instruction set is the set of operations which the CPU can perform while a program is executing ("at run-time"), and is strictly determined by the microcomputer hardware design.

The assembly language is a standard (though more-or-less arbitrary) set of symbols including the instruction set mnemonics, but with additional features which further simplify the program design process. For example, ASM51 has controls for creating and formatting a program listing, and a number of directives for allocating variable storage and inserting arbitrary bytes of data into the object code for creating tables of constants.

In addition, ASM51 can perform sophisticated mathematical operations, computing addresses or evaluating arithmetic expressions to relieve the programmer from this drudgery. However, these calculations can only use information known at "assembly time."

For example, the 8051 performs arithmetic calculations at run-time, eight bits at a time. ASM51 can do similar operations 16 bits at a time. The 8051 can only do one simple step per instruction, while ASM51 can perform complex calculations in each line of source code. However, the operations performed by the assembler may only use parameter values fixed at assembly-time, not variables whose values are unknown until program execution begins.

For example, when the assembly language source line,

ADD A,#(LOOP_COUNT + 1) * 3

is assembled, ASM51 will find the value of the previously-defined constant "LOOP_COUNT" in an internal symbol table, increment the value, multiply the sum by three, and (assuming it is between -256 and 255 inclusive) truncate the product to eight bits. When this instruction is executed, the 8051 ALU will just add that resulting constant to the accumulator.

Some similar differences exist to distinguish number system ("radix") specifications. The 8051 does all computations in binary (though there are provisions for then converting the result to decimal form). In the course of writing a program, though, it may be more convenient to specify constants using some other radix, such as base 10. On other occasions, it is desirable to specify the ASCII code for some character or string of characters without referring to tables. ASM51 allows several representations for constants, which are converted to binary as each instruction is assembled.

For example, binary numbers are represented in the

assembly language by a series of ones and zeros (naturally), followed by the letter "B" (for Binary); octal numbers as a series of octal digits (0-7) followed by the letter "O" (for Octal) or "Q" (which doesn't stand for anything, but looks sort of like an "O" and is less likely to be confused with a zero).

Hexadecimal numbers are represented by a series of hexadecimal digits (0-9,A-F), followed by (you guessed it) the letter "H." A "hex" number must begin with a decimal digit; otherwise it would look like a user-defined symbol (to be discussed later). A "dummy" leading zero may be inserted before the first digit to meet this constraint. The character string "BACH" could be a legal label for a Baroque music synthesis routine; the string "0BACH" is the hexadecimal constant BAC₁₆. This is a case where adding 0 makes a big difference.

Decimal numbers are represented by a sequence of decimal digits, optionally followed by a "D." If a number has no suffix, it is assumed to be decimal—so it had better not contain any non-decimal digits. "0BAC" is not a legal representation for anything.

When an ASCII code is needed in a program, enclose the desired character between two apostrophes (as in '#') and the assembler will convert it to the appropriate code (in this case 23H). A string of characters between apostrophes is translated into a series of constants; 'BACH' becomes 42H, 41H, 43H, 48H.

These same conventions are used throughout the associated Intel documentation. Table 2 illustrates some of the different number formats.

2. ARCHITECTURE AND ORGANIZATION

Figure 3 blocks out the MCS-51™ internal organization. Each microcomputer combines a Central Processing Unit, two kinds of memory (data RAM plus program ROM or EPROM), Input/Output ports, and the mode,

Table 2. Notations Used to Represent Numbers

Bit Pattern	Binary	Octal	Hexa-Decimal	Decimal	Signed Decimal
0 0 0 0 0 0 0 0	0B	0Q	00H	0	0
0 0 0 0 0 0 0 1	1B	1Q	01H	1	+1
...
0 0 0 0 0 1 1 1	111B	7Q	07H	7	+7
0 0 0 0 1 0 0 0	1000B	10Q	08H	8	+8
0 0 0 0 1 0 0 1	1001B	11Q	09H	9	+9
0 0 0 0 1 0 1 0	1010B	12Q	0AH	10	+10
...
0 0 0 0 1 1 1 1	1111B	17Q	0FH	15	+15
0 0 0 1 0 0 0 0	10000B	20Q	10H	16	+16
...
0 1 1 1 1 1 1 1	1111111B	177Q	7FH	127	+127
1 0 0 0 0 0 0 0	10000000B	200Q	80H	128	-128
1 0 0 0 0 0 0 1	10000001B	201Q	81H	129	-127
...
1 1 1 1 1 1 1 0	11111110B	376Q	0FEH	254	-2
1 1 1 1 1 1 1 1	11111111B	377Q	0FFH	255	-1

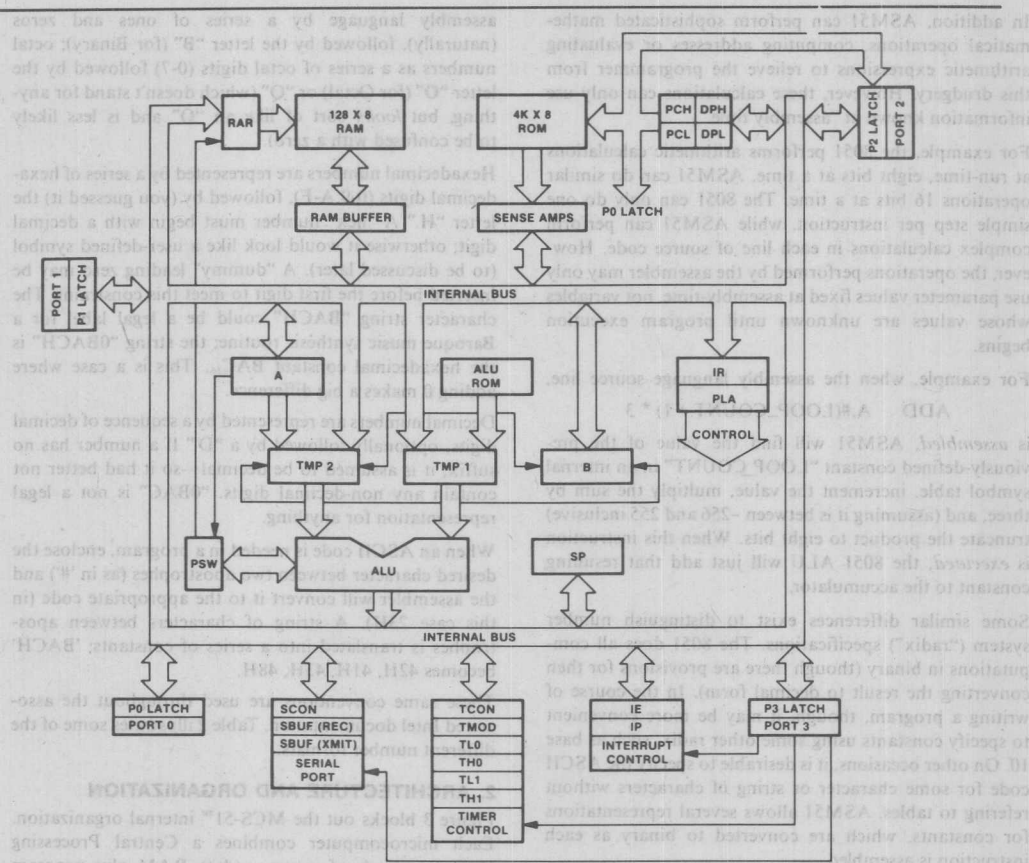


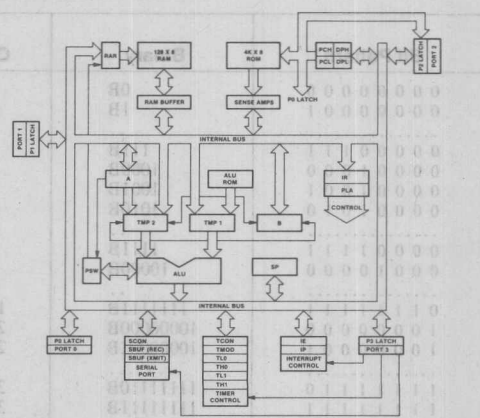
Figure 3. Block Diagram of 8051 Internal Structure

status, and data registers and random logic needed for a variety of peripheral functions. These elements communicate through an eight-bit data bus which runs throughout the chip, somewhat akin to indoor plumbing. This bus is buffered to the outside world through an I/O port when memory or I/O expansion is desired.

Let's summarize what each block does; later chapters dig into the CPU's instruction set and the peripheral registers in much greater detail.

Central Processing Unit

The CPU is the "brains" of the microcomputer, reading the user's program and executing the instructions stored therein. Its primary elements are an eight-bit Arithmetic/Logic Unit with associated registers A, B, PSW, and SP, and the sixteen-bit Program Counter and "Data Pointer" registers.



Arithmetic Logic Unit

The ALU can perform (as the name implies) arithmetic and logic functions on eight-bit variables. The former include basic addition, subtraction, multiplication, and division; the latter include the logical operations AND, OR, and Exclusive-OR, as well as rotate, clear, complement, and so forth. The ALU also makes conditional branching decisions, and provides data paths and temporary registers used for data transfers within the system. Other instructions are built up from these primitive functions: the addition capability can increment registers or automatically compute program destination addresses; subtraction is also used in decrementing or comparing the magnitude of two variables.

These primitive operations are automatically cascaded and combined with dedicated logic to build complex instructions such as incrementing a sixteen-bit register pair. To execute one form of the compare instruction, for example, the 8051 increments the program counter three times, reads three bytes of program memory, computes a register address with logical operations, reads internal data memory twice, makes an arithmetic comparison of two variables, computes a sixteen-bit destination address, and decides whether or not to make a branch—all in two microseconds!

An important and unique feature of the MCS-51 architecture is that the ALU can also manipulate one-bit as well as eight-bit data types. Individual bits may be set, cleared, or complemented, moved, tested, and used in logic computations. While support for a more primitive data type may initially seem a step backwards in an era of increasing word length, it makes the 8051 especially well suited for controller-type applications. Such algorithms inherently involve Boolean (true/false) input and output variables, which were heretofore difficult to implement with standard microprocessors. These features are collectively referred to as the MCS-51™ "Boolean Processor," and are described in the so-named chapter to come.

Thanks to this powerful ALU, the 8051 instruction set fares well at both real-time control and data intensive algorithms. A total of 51 separate operations move and manipulate three data types: Boolean (1-bit), byte (8-bit), and address (16-bit). All told, there are eleven addressing modes—seven for data, four for program sequence control (though only eight are used by more than just a few specialized instructions). Most operations allow several addressing modes, bringing the total number of instructions (operation/addressing mode combinations) to 111, encompassing 255 of the 256 possible eight-bit instruction opcodes.

Instruction Set Overview

Table 4 lists these 111 instructions classified into five groups:

- Arithmetic Operations
- Logical Operations for Byte Variables
- Data Transfer Instructions
- Boolean Variable Manipulation
- Program Branching and Machine Control

MCS-48™ programmers perusing Table 4 will notice the absence of special categories for Input/Output, Timer/Counter, or Control instructions. These functions are all still provided (and indeed many new functions are added), but as special cases of more generalized operations in other categories. To explicitly list all the useful instructions involving I/O and peripheral registers would require a table approximately four times as long.

Observant readers will also notice that all of the 8048's page-oriented instructions (conditional jumps, JMPP, MOVP, MOVP3) have been replaced with corresponding but non-paged instructions. The 8051 instruction set is entirely non-page-oriented. The MCS-48™ "MOVP" instruction replacement and all conditional jump instructions operate relative to the program counter, with the actual jump address computed by the CPU during instruction execution. The "MOVP3" and "JMPP" replacements are now made relative to another sixteen-bit register, which allows the effective destination to be anywhere in the program memory space, regardless of where the instruction itself is located. There are even three-byte jump and call instructions allowing the destination to be anywhere in the 64K program address space.

The instruction set is designed to make programs efficient both in terms of code size and execution speed. No instruction requires more than three bytes of program memory, with the majority requiring only one or two bytes. Virtually all instructions execute in either one or two instruction cycles—one or two microseconds with a 12-MHz crystal—with the sole exceptions (multiply and divide) completing in four cycles.

Many instructions such as arithmetic and logical functions or program control, provide both a short and a long form for the same operation, allowing the programmer to optimize the code produced for a specific application. The 8051 usually fetches two instruction bytes per instruction cycle, so using a shorter form can lead to faster execution as well.

For example, any byte of RAM may be loaded with a constant with a three-byte, two-cycle instruction, but the commonly used "working registers" in RAM may be initialized in one cycle with a two-byte form. Any bit anywhere on the chip may be set, cleared, or complemented by a single three-byte logical instruction using two cycles. But critical control bits, I/O pins, and software flags may be controlled by two-byte, single cycle instructions. While three-byte jumps and calls can "go anywhere" in program memory, nearby sections of code may be reached by shorter relative or absolute versions.

AFN-01502A-09

(MSB)				(LSB)			
CY	AC	F0	RS1	RS0	OV	—	P

Symbol Position Name and Significance

CY	PSW.7	Carry flag. Set/cleared by hardware or software during certain arithmetic and logical instructions.
AC	PSW.6	Auxiliary Carry flag. Set/cleared by hardware during addition or subtraction instructions to indicate carry or borrow out of bit 3.
F0	PSW.5	Flag 0 Set/cleared/tested by software as a user-defined status flag.
RS1	PSW.4	Register bank Select control bits 1 & 0. Set/cleared by software to determine working register bank (see Note).
RS	PSW.3	

Symbol Position Name and Significance

OV	PSW.2	Overflow flag. Set/cleared by hardware during arithmetic instructions to indicate overflow conditions.
—	PSW.1	(reserved)
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the accumulator, i.e., even parity.

Note— the contents of (RS1, RS0) enable the working register banks as follows:

(0,0) — Bank 0	(00H-07H)
(0,1) — Bank 1	(08H-0FH)
(1,0) — Bank 2	(10H-17H)
(1,1) — Bank 3	(18H-1FH)

Figure 4. PSW—Program Status Word Organization

A significant side benefit of an instruction set more powerful than those of previous single-chip microcomputers is that it is easier to generate applications-oriented software. Generalized addressing modes for byte and bit instructions reduce the number of source code lines written and debugged for a given application. This leads in turn to proportionately lower software costs, greater reliability, and faster design cycles.

Accumulator and PSW

The 8051, like its 8048 predecessor, is primarily an accumulator-based architecture: an eight-bit register called the accumulator ("A") holds a source operand and receives the result of the arithmetic instructions (addition, subtraction, multiplication, and division). The accumulator can be the source or destination for logical operations and a number of special data movement instructions, including table look-ups and external RAM expansion. Several functions apply exclusively to the accumulator: rotates, parity computation, testing for zero, and so on.

Many instructions implicitly or explicitly affect (or are affected by) several status flags, which are grouped together to form the Program Status Word shown in Figure 4.

(The period within entries under the Position column is called the "dot operator," and indicates a particular bit position within an eight-bit byte. "PSW.5" specifies bit 5 of the PSW. Both the documentation and ASM51 use this notation.)

The most "active" status bit is called the carry flag (abbreviated "C"). This bit makes possible multiple precision arithmetic operations including addition, subtraction,

and rotates. The carry also serves as a "Boolean accumulator" for one-bit logical operations and bit manipulation instructions. The overflow flag (OV) detects when arithmetic overflow occurs on signed integer operands, making two's complement arithmetic possible. The parity flag (P) is updated after every instruction cycle with the even-parity of the accumulator contents.

The CPU does not control the two register-bank select bits, RS1 and RS0. Rather, they are manipulated by software to enable one of the four register banks. The usage of the PSW flags is demonstrated in the Instruction Set chapter of this Note.

Even though the architecture is accumulator-based, provisions have been made to bypass the accumulator in common instruction situations. Data may be moved from any location on-chip to any register, address, or indirect address (and vice versa), any register may be loaded with a constant, etc., all without affecting the accumulator. Logical operations may be performed against registers or variables to alter fields of bits—without using or affecting the accumulator. Variables may be incremented, decremented, or tested without using the accumulator. Flags and control bits may be manipulated and tested without affecting anything else.

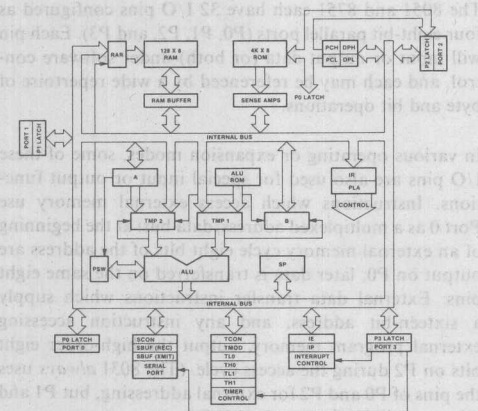
Other CPU Registers

A special eight-bit register ("B") serves in the execution of the multiply and divide instructions. This register is used in conjunction with the accumulator as the second input operand and to return eight-bits of the result.

The MCS-51 family processors include a hardware stack within internal RAM, useful for subroutine linkage,

passing parameters between routines, temporary variable storage, or saving status during interrupt service routines. The Stack Pointer (SP) is an eight-bit pointer register which indicates the address of the last byte pushed onto the stack. The stack pointer is automatically incremented or decremented on all push or pop instructions and all subroutine calls and returns. In theory, the stack in the 8051 may be up to a full 128 bytes deep. (In practice, even simple programs would use a handful of RAM locations for pointers, variables, and so forth—reducing the stack depth by that number.) The stack pointer defaults to 7 on reset, so that the stack will start growing up from location 8, just like in the 8048. By altering the pointer contents the stack may be relocated anywhere within internal RAM.

Finally, a 16-bit register called the data pointer (DPTR) serves as a base register in indirect jumps, table look-up instructions, and external data transfers. The high- and low-order halves of the data pointer may be manipulated as separate registers (DPH and DPL, respectively) or together using special instructions to load or increment all sixteen bits. Unlike the 8048, look-up tables can therefore start anywhere in program memory and be of arbitrary length.



Memory Spaces

Program memory is separate and distinct from data memory. Each memory type has a different addressing mechanism, different control signals, and a different function.

The program memory array (ROM or EPROM), like an elephant, is extremely large and never forgets information, even when power is removed. Program memory is used for information needed each time power is applied: initialization values, calibration constants, keyboard layout tables, etc., as well as the program itself. The program memory has a sixteen-bit address bus; its elements

are addressed using the Program Counter or instructions which generate a sixteen-bit address.

To stretch our analogy just a bit, data memory is like a mouse: it is smaller and therefore quicker than program memory, and it goes into a random state when electrical power is applied. On-chip data RAM is used for variables which are determined or may change while the program is running.

A computer spends most of its time manipulating variables, not constants, and a relatively small number of variables at that. Since eight-bits is more than sufficient to uniquely address 128 RAM locations, the on-chip RAM address register is only one byte wide. In contrast to the program memory, data memory accesses need a single eight-bit value—a constant or another variable—to specify a unique location. Since this is the basic width of the ALU and the different memory types, those resources can be used by the addressing mechanisms, contributing greatly to the computer's operating efficiency.

The partitioning of program and data memory is extended to off-chip memory expansion. Each may be added independently, and each uses the same address and data busses, but with different control signals. External program memory is gated onto the external data bus by the PSEN (Program Store Enable) control output, pin 29. External data memory is read onto the bus by the RD output, pin 17, and written with data supplied from the microcomputer by the WR output, pin 16. (There is no control pin to write external program ROM, which is by definition Read Only.) While both types may be expanded to up to 64K bytes, the external data memory may optionally be expanded in 256 byte "pages" to preserve the use of P2 as an I/O port. This is useful with a relatively small expansion RAM (such as the Intel® 8155) or for addressing external peripherals.

Single-chip controller programs are finalized during the project design cycle, and are not modified after production. Intel's single-chip microcomputers are not "von Neumann" architectures common among main-frame and mini-computer systems: the MCS-51™ processor data memory—on-chip and external—may not be used for program code. Just as there is no write-control signal for program memory, there is no way for the CPU to execute instructions out of RAM. In return, this concession allows an architecture optimized for efficient controller applications: a large, fixed program located in ROM, a hundred or so variables in RAM, and different methods for efficiently addressing each.

(Von Neumann machines are helpful for software development and debug. An 8051 system could be modified to have a single off-chip memory space by gating together the two memory-read controls (PSEN and RD) with a two-input AND gate (Figure 5). The CPU could then write data into the common memory array using WR and

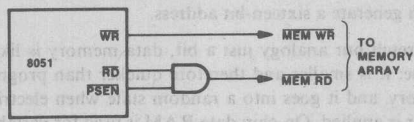


Figure 5. Combining External Program and Data Memory Arrays

external data transfer instructions, and read instructions or data with the AND gate output and data transfer or program memory look-up instructions.)

In addition to the memory arrays, there is (yet) another (albeit sparsely populated) physical address space. Connected to the internal data bus are a score of special-purpose eight-bit registers scattered throughout the chip. Some of these—B, SP, PSW, DPH, and DPL—have been discussed above. Others—I/O ports and peripheral function registers—will be introduced in the following sections. Collectively, these registers are designated as the “special-function register” address space. Even the accumulator is assigned a spot in the special-function register address space for additional flexibility and uniformity.

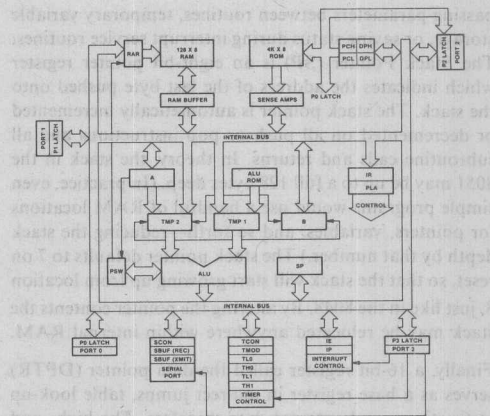
Thus, the MCS-51™ architecture supports several distinct “physical” address spaces, functionally separated at the hardware level by different addressing mechanisms, read and write control signals, or both:

- On-chip program memory;
- On-chip data memory;
- Off-chip program memory;
- Off-chip data memory;
- On-chip special-function registers.

What the *programmer sees*, though, are “logical” address spaces. For example, as far as the programmer is concerned, there is only one type of program memory, 64K bytes in length. The fact that it is formed by combining on- and off-chip arrays (split 4K/60K on the 8051 and 8751) is “invisible” to the programmer; the CPU automatically fetches each byte from the appropriate array, based on its address.

(Presumably, future microcomputers based on the MCS-51™ architecture may have a different physical split, with more or less of the 64K total implemented on-chip. Using the MCS-48™ family as a precedent, the 8048's 4K potential program address space was split 1K/3K between on- and off-chip arrays; the 8049's was split 2K/2K.)

Why go into such tedious details about address spaces? The logical addressing modes are described in the Instruction Set chapter in terms of physical address spaces. Understanding their differences now will pay off in understanding and using the chips later.



(MSB)				(LSB)			
RD	WR	T1	T0	INT1	INT0	TxD	RxD

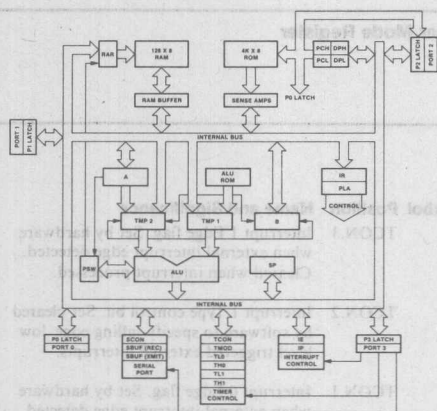
Symbol Position Name and Significance

RD	P3.7	Read data control output. Active low pulse generated by hardware when external data memory is read.
WR	P3.6	Write data control output. Active low pulse generated by hardware when external data memory is written.
T1	P3.5	Timer/counter 1 external input or test pin.
T0	P3.4	Timer/counter 0 external input or test pin.

Symbol Position Name and Significance

INT1	P3.3	Interrupt 1 input pin. Low-level or falling-edge triggered.
INT0	P3.2	Interrupt 0 input pin. Low-level or falling-edge triggered.
TxD	P3.1	Transmit Data pin for serial port in UART mode. Clock output in shift register mode.
RxD	P3.0	Receive Data pin for serial port in UART mode. Data I/O pin in shift register mode.

Figure 6. P3—Alternate Special Functions of Port 3



Special Peripheral Functions

There are a few special needs common among control-oriented computer systems:

- keeping track of elapsed real-time;
- maintaining a count of signal transitions;
- measuring the precise width of input pulses;
- communicating with other systems or people;
- closely monitoring asynchronous external events.

Until now, microprocessor systems needed peripheral chips such as timer/counters, USARTs, or interrupt controllers to meet these needs. The 8051 integrates all of these capabilities on-chip!

Timer/Counters

There are two sixteen-bit multiple-mode Timer/Counters on the 8051, each consisting of a "High" byte (corresponding to the 8048 "T" register) and a low byte (similar to the 8048 prescaler, with the additional flexibility of being

software-accessible). These registers are called, naturally enough, TH0, TL0, TH1, and TL1. Each pair may be independently software programmed to any of a dozen modes with a mode register designated TMOD (Figure 7), and controlled with register TCON (Figure 8).

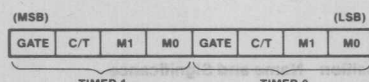
The timer modes can be used to measure time intervals, determine pulse widths, or initiate events, with one-micro-second resolution, up to a maximum interval of 65,536 instruction cycles (over 65 milliseconds). Longer delays may easily be accumulated through software. Configured as a counter, the same hardware will accumulate external events at frequencies from D.C. to 500 KHz, with up to sixteen bits of precision.

Serial Port Interface

Each microcomputer contains a high-speed, full-duplex, serial port which is software programmable to function in four basic modes: shift-register I/O expander, 8-bit UART, 9-bit UART, or interprocessor communications link. The UART modes will interface with standard I/O devices (e.g. CRTs, teletypewriters, or modems) at data rates from 122 baud to 31 kilobaud. Replacing the standard 12 MHz crystal with a 10.7 MHz crystal allows 110 baud. Even or odd parity (if desired) can be included with simple bit-handling software routines. Inter-processor communications in distributed systems takes place at 187 kilobaud with hardware for automatic address/data message recognition. Simple TTL or CMOS shift registers provide low-cost I/O expansion at a super-fast 1 Megabaud. The serial port operating modes are controlled by the contents of register SCON (Figure 9).

Interrupt Capability and Control

(Interrupt capability is generally considered a CPU function. It is being introduced here since, from an applications point of view, interrupts relate more closely to peripheral and system interfacing.)



GATE Gating control. When set, Timer/counter "x" is enabled only while "INTx" pin is high and "TRx" control bit is set. When cleared, timer/counter is enabled whenever "TRx" control bit is set.

C/T Timer or Counter Selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from "Tx" input pin).

M1	M0	Operating Mode
0	0	MCS-48 Timer. "TLx" serves as five-bit prescaler.
0	1	16-bit timer/counter. "THx" and "TLx" are cascaded; there is no prescaler.
1	0	8-bit auto-reload timer counter. "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
1	1	(Timer 0) TL0 is an eight-bit timer counter controlled by the standard Timer 0 control bits. TH0 is an eight-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer counter 1 stopped.

Figure 7. TMOD—Timer/Counter Mode Register



Symbol	Position	Name and Significance
TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.

Symbol	Position	Name and Significance
IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge low level triggered external interrupts.
IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT0	TCON.0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge low level triggered external interrupts.

Figure 8. TCON—Timer/Counter Control/Status Register

(MSB)				(LSB)			
SM0	SM1	SM2	REN	TB8	RB8	TI	RI
Symbol	Position	Name and Significance					
SM0	SCON.7	Serial port Mode control bit 0. Set/cleared by software (see note).					
SM1	SCON.6	Serial port Mode control bit 1. Set/cleared by software (see note).					
SM2	SCON.5	Serial port Mode control bit 2. Set by software to disable reception of frames for which bit 8 is zero.					
REN	SCON.4	Receiver Enable control bit. Set/cleared by software to enable/disable serial data reception.					
TB8	SCON.3	Transmit Bit 8. Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.					
Symbol	Position	Name and Significance					
RB8	SCON.2	Receive Bit 8. Set/cleared by hardware to indicate state of ninth data bit received.					
TI	SCON.1	Transmit Interrupt flag. Set by hardware when byte transmitted. Cleared by software after servicing.					
RI	SCON.0	Received Interrupt flag. Set by hardware when byte received. Cleared by software after servicing.					
Note—		the state of (SM0,SM1) selects: (0,0)—Shift register I/O expansion. (0,1)—8 bit UART, variable data rate. (1,0)—9 bit UART, fixed data rate. (1,1)—9 bit UART, variable data rate.					

Figure 9. SCON—Serial Port Control/Status Register

These peripheral functions allow special hardware to monitor real-time signal interfacing without bothering the CPU. For example, imagine serial data is arriving from one CRT while being transmitted to another, and one timer/counter is tallying high-speed input transitions while the other measures input pulse widths. During all of this the CPU is thinking about something else.

But how does the CPU know when a reception, transmission, count, or pulse is finished? The 8051 programmer can choose from three approaches.

TCON and SCON contain status bits set by the hardware when a timer overflows or a serial port operation is completed. The first technique reads the control register into the accumulator, tests the appropriate bit, and does a conditional branch based on the result. This "polling" scheme (typically a three-instruction sequence though additional instructions to save and restore the accumulator may sometimes be needed) will surely be familiar to programmers used to multi-chip microcomputer systems and peripheral controller chips. This process is rather cumbersome, especially when monitoring multiple peripherals.

As a second approach, the 8051 can perform a conditional branch based on the state of any control or status bit or input pin in a single instruction; a four instruction sequence could poll the four simultaneous happenings mentioned above in just eight microseconds.

Unfortunately, the CPU must still drop what it's doing to test these bits. A manager cannot do his own work well if he is continuously monitoring his subordinates; they should interrupt him (or her) only when they need attention or guidance. So it is with machines: ideally, the CPU would not have to worry about the peripherals until they require servicing. At that time, it would postpone the

background task long enough to handle the appropriate device, then return to the point where it left off.

This is the basis of the third and generally optimal solution, hardware interrupts. The 8051 has five interrupt sources: one from the serial port when a transmission or reception is complete, two from the timers when overflows occur, and two from input pins INT0 and INT1. Each source may be independently enabled or disabled to allow polling on some sources or at some times, and each may be classified as high or low priority. A high priority source can interrupt a low priority service routine; the manager's boss can interrupt conferences with subordinates. These options are selected by the interrupt enable and priority control registers, IE and IP (Figures 10 and 11).

Each source has a particular program memory address associated with it (Table 3), starting at 0003H (as in the 8048) and continuing at eight-byte intervals. When an event enabled for interrupts occurs the CPU automatically executes an internal subroutine call to the corresponding address. A user subroutine starting at this location (or jumped to from this location) then performs the instructions to service that particular source. After completing the interrupt service routine, execution returns to the background program.

Table 3. 8051 Interrupt Sources and Service Vectors

Interrupt Source	Service Routine Starting Address
(Reset)	0000H
External 0	0003H
Timer/Counter 0	000BH
External 1	0013H
Timer/Counter 1	001BH
Serial Port	0023H

AFN-01502A-15

(MSB)				(LSB)			
EA	—	—	ES	ET1	EX1	ET0	EX0

Symbol Position Name and Significance

EA	IE.7	Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4-IE.0.
—	IE.6	(reserved)
—	IE.5	(reserved)
ES	IE.4	Enable Serial port control bit. Set/cleared by software to enable/disable interrupts from TI or RI flags.
ET1	IE.3	Enable Timer 1 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 1.

(MSB)				(LSB)			
IE7	IE6	IE5	IE4	IE3	IE2	IE1	IE0

Symbol Position Name and Significance

EX1	IE.2	Enable External interrupt 1 control bit. Set cleared by software to enable/disable interrupts from INT1.
ET0	IE.1	Enable Timer 0 control bit. Set cleared by software to enable/disable interrupts from timer counter 0.
EX0	IE.0	Enable External interrupt 0 control bit. Set cleared by software to enable/disable interrupts from INT0.

Figure 10. IE—Interrupt Enable Register

(MSB)				(LSB)			
—	—	—	PS	PT1	PX1	PT0	PX0

Symbol Position Name and Significance

—	IP.7	(reserved)
—	IP.6	(reserved)
—	IP.5	(reserved)
PS	IP.4	Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.
PT1	IP.3	Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.

Symbol Position Name and Significance

PX1	IP.2	External interrupt 1 Priority control bit. Set cleared by software to specify high/low priority interrupts for INT1.
PT0	IP.1	Timer 0 Priority control bit. Set cleared by software to specify high/low priority interrupts for timer counter 0.
PX0	IP.0	External interrupt 0 Priority control bit. Set cleared by software to specify high/low priority interrupts for INT0.

Figure 11. IP—Interrupt Priority Control Register

Table 3. 8051 Interrupt Sources and Service Vectors

Interrupt Source	Service Routine Starting Address
Serial Port	0023H
Timer/Counter 1	001BH
External 1	0013H
Timer/Counter 0	000BH
External 0	0003H
(Reserved)	0000H

Table 4. MCS-51™ Instruction Set Description

ARITHMETIC OPERATIONS				DATA TRANSFER (cont.)			
Mnemonic	Description	Byte	Cyc	Mnemonic	Description	Byte	Cyc
ADD A,Rn	Add register to Accumulator	1	1	MOVC A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
ADD A,direct	Add direct byte to Accumulator	2	1	MOVC A,@A+PC	Move Code byte relative to PC to A	1	2
ADD A,@Ri	Add indirect RAM to Accumulator	1	1	MOVX A,@Ri	Move External RAM (8-bit addr) to A	1	2
ADD A,#data	Add immediate data to Accumulator	2	1	MOVX A,DPTR	Move External RAM (16-bit addr) to A	1	2
ADDC A,Rn	Add register to Accumulator with Carry	1	1	MOVX @Ri,A	Move A to External RAM (8-bit addr)	1	2
ADDC A,direct	Add direct byte to A with Carry flag	2	1	MOVX @DPTR,A	Move A to External RAM (16-bit addr)	1	2
ADDC A,@Ri	Add indirect RAM to A with Carry flag	1	1	PUSH direct	Push direct byte onto stack	2	2
ADDC A,#data	Add immediate data to A with Carry flag	2	1	POP direct	Pop direct byte from stack	2	2
SUBB A,Rn	Subtract register from A with Borrow	1	1	XCH A,Rn	Exchange register with Accumulator	1	1
SUBB A,direct	Subtract direct byte from A with Borrow	2	1	XCH A,direct	Exchange direct byte with Accumulator	2	1
SUBB A,@Ri	Subtract indirect RAM from A w Borrow	1	1	XCH A,@Ri	Exchange indirect RAM with A	1	1
SUBB A,#data	Subtract immediate data from A w Borrow	2	1	XCHD A,@Ri	Exchange low-order Digit ind. RAM w A	1	1
INC A	Increment Accumulator	1	1	BOOLEAN VARIABLE MANIPULATION			
INC Rn	Increment register	1	1	Mnemonic	Description	Byte	Cyc
INC direct	Increment direct byte	2	1	CLR C	Clear Carry flag	1	1
INC @Ri	Increment indirect RAM	1	1	CLR bit	Clear direct bit	2	1
DEC A	Decrement Accumulator	1	1	SETB C	Set Carry flag	1	1
DEC Rn	Decrement register	1	1	SETB bit	Set direct Bit	2	1
DEC direct	Decrement direct byte	2	1	CPL C	Complement Carry flag	1	1
DEC @Ri	Decrement indirect RAM	1	1	CPL bit	Complement direct bit	2	1
INC DPTR	Increment Data Pointer	1	2	ANL C,bit	AND direct bit to Carry flag	2	2
MUL AB	Multiply A & B	1	4	ANL C,bit	AND complement of direct bit to Carry	2	2
DIV AB	Divide A by B	1	4	ORL C,bit	OR direct bit to Carry flag	2	2
DA A	Decimal Adjust Accumulator	1	1	ORL C,bit	OR complement of direct bit to Carry	2	2
				MOV C,bit	Move direct bit to Carry flag	2	1
				MOV bit,C	Move Carry flag to direct bit	2	2
LOGICAL OPERATIONS				PROGRAM AND MACHINE CONTROL			
Mnemonic	Destination	Byte	Cyc	Mnemonic	Description	Byte	Cyc
ANL A,Rn	AND register to Accumulator	1	1	ACALL addr11	Absolute Subroutine Call	2	2
ANL A,direct	AND direct byte to Accumulator	2	1	LCALL addr16	Long Subroutine Call	3	2
ANL A,@Ri	AND indirect RAM to Accumulator	1	1	RET	Return from subroutine	1	2
ANL A,#data	AND immediate data to Accumulator	2	1	RETI	Return from interrupt	1	2
ANL direct,A	AND Accumulator to direct byte	2	1	AJMP addr11	Absolute Jump	2	2
ANL direct,#data	AND immediate data to direct byte	3	2	LJMP addr16	Long Jump	3	2
ORL A,Rn	OR register to Accumulator	1	1	SJMP rel	Short Jump (relative addr)	2	2
ORL A,direct	OR direct byte to Accumulator	2	1	JMP @A+DPTR	Jump indirect relative to the DPTR	1	2
ORL A,@Ri	OR indirect RAM to Accumulator	1	1	JZ rel	Jump if Accumulator is Zero	2	2
ORL A,#data	OR immediate data to Accumulator	2	1	JNZ rel	Jump if Accumulator is Not Zero	2	2
ORL direct,A	OR Accumulator to direct byte	2	1	JC rel	Jump if Carry flag is set	2	2
ORL direct,#data	OR immediate data to direct byte	3	2	JNC rel	Jump if No Carry flag	2	2
XRL A,Rn	Exclusive-OR register to Accumulator	1	1	JB bit,rel	Jump if direct Bit set	3	2
XRL A,direct	Exclusive-OR direct byte to Accumulator	2	1	JNB bit,rel	Jump if direct Bit Not set	3	2
XRL A,@Ri	Exclusive-OR indirect RAM to A	1	1	JBC bit,rel	Jump if direct Bit is set & Clear bit	3	2
XRL A,#data	Exclusive-OR immediate data to A	2	1	CJNE A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
XRL direct,A	Exclusive-OR Accumulator to direct byte	2	1	CJNE A,#data,rel	Comp. immed. to A & Jump if Not Equal	3	2
XRL direct,#data	Exclusive-OR immediate data to direct	3	2	CJNE Rn,#data,rel	Comp. immed. to reg. & Jump if Not Equal	3	2
CLR A	Clear Accumulator	1	1	CJNE @Ri,#data,rel	Comp. indirect to ind. & Jump if Not Equal	3	2
CPL A	Complement Accumulator	1	1	DJNZ Rn,rel	Decrement register & Jump if Not Zero	2	2
RL A	Rotate Accumulator Left	1	1	DJNZ direct,rel	Decrement direct & Jump if Not Zero	3	2
RLC A	Rotate A Left through the Carry flag	1	1	NOP	No operation	1	1
RR A	Rotate Accumulator Right	1	1				
RRC A	Rotate A Right through Carry flag	1	1				
SWAP A	Swap nibbles within the Accumulator	1	1				
DATA TRANSFER				Notes on data addressing modes:			
Mnemonic	Description	Byte	Cyc	Rn	Working register R0-R7		
MOV A,Rn	Move register to Accumulator	1	1	direct	128 internal RAM locations, any I/O port, control or status register		
MOV A,direct	Move direct byte to Accumulator	2	1	@Ri	Indirect internal RAM location addressed by register R0 or R1		
MOV A,@Ri	Move indirect RAM to Accumulator	1	1	#data	8-bit constant included in instruction		
MOV A,#data	Move immediate data to Accumulator	2	1	#data16	16-bit constant included as bytes 2 & 3 of instruction		
MOV Rn,A	Move Accumulator to register	1	1	bit	128 software flags, any I/O pin, control or status bit		
MOV Rn,direct	Move direct byte to register	2	2	Notes on program addressing modes:			
MOV Rn,#data	Move immediate data to register	2	1	addr16	Destination address for LCALL & LJMP may be anywhere within the 64-Kilobyte program memory address space.		
MOV direct,A	Move Accumulator to direct byte	2	1	addr11	Destination address for ACALL & AJMP will be within the same 2-Kilobyte page of program memory as the first byte of the following instruction.		
MOV direct,Rn	Move register to direct byte	2	2	rel	SJMP and all conditional jumps include an 8-bit offset byte. Range is +127-128 bytes relative to first byte of the following instruction.		
MOV direct,direct	Move direct byte to direct	3	2				
MOV direct,@Ri	Move indirect RAM to direct byte	2	2				
MOV direct,#data	Move immediate data to direct byte	3	2				
MOV @Ri,A	Move Accumulator to indirect RAM	1	1				
MOV @Ri,direct	Move direct byte to indirect RAM	2	2				
MOV @Ri,#data	Move immediate data to indirect RAM	2	1				
MOV DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2				

3. INSTRUCTION SET AND ADDRESSING MODES

The 8051 instruction set is extremely regular, in the sense that most instructions can operate with variables from several different physical or logical address spaces. Before getting deeply enmeshed in the instruction set proper, it is important to understand the details of the most common data addressing modes. Whereas Table 4 summarizes the instructions set broken down by functional

group, this chapter starts with the addressing mode classes and builds to include the related instructions.

Data Addressing Modes

MCS-51 assembly language instructions consist of an operation mnemonic and zero to three operands separated by commas. In two operand instructions the destination is specified first, then the source. Many byte-wide data

operations (such as ADD or MOV) inherently use the accumulator as a source operand and/or to receive the result. For the sake of clarity the letter "A" is specified in the source or destination field in all such instructions. For example, the instruction,

```
ADD A,<source>
```

will add the variable<source>to the accumulator, leaving the sum in the accumulator.

The operand designated "<source>" above may use any of four common logical addressing modes:

- Register—one of the working registers in the currently enabled bank.
- Direct—an internal RAM location, I/O port, or special-function register.
- Register-indirect—an internal RAM location, pointed to by a working register.
- Immediate data—an eight-bit constant incorporated into the instruction.

The first three modes provide access to the internal RAM and Hardware Register address spaces, and may therefore be used as source or destination operands; the last mode accesses program memory and may be a source operand only.

(It is hard to show a "typical application" of any instruction without involving instructions not yet described. The following descriptions use only the self-explanatory ADD and MOV instructions to demonstrate how the four addressing modes are specified and used. Subsequent examples will become increasingly complex.)

Register Addressing

The 8051 programmer has access to eight "working registers," numbered R0-R7. The least-significant three-bits of the instruction opcode indicate one register within this logical address space. Thus, a function code and operand address can be combined to form a short (one byte) instruction (Figure 12.a).

The 8051 assembly language indicates register addressing with the symbol Rn (where n is from 0 to 7) or with a symbolic name previously defined as a register by the EQUate or SET directives. (For more information on assembler directives see the Macro Assembler Reference Manual.)

Example 1—Adding Two Registers Together

```
REGADR ADD CONTENTS OF REGISTER 1
      TO CONTENTS OF REGISTER 0
REGADR MOV A,R0
      ADD A,R1
      MOV RO,A
```

There are four such banks of working registers, only one of which is active at a time. Physically, they occupy the first 32 bytes of on-chip data RAM (addresses 0-1FH). PSW bits 4 and 3 determine which bank is active. A

hardware reset enables register bank 0; to select a different bank the programmer modifies PSW bits 4 and 3 accordingly.

Example 2—Selecting Alternate Memory Banks

```
MOV PSW,#00010000B ;SELECT BANK 2
```

Register addressing in the 8051 is the same as in the 8048 family, with two enhancements: there are four banks rather than one or two, and 16 instructions (rather than 12) can access them.

Direct Byte Addressing

Direct addressing can access any on-chip variable or hardware register. An additional byte appended to the opcode specifies the location to be used (Figure 12.b).

Depending on the highest order bit of the direct address byte, one of two physical memory spaces is selected. When the direct address is between 0 and 127 (00H-7FH) one of the 128 low-order on-chip RAM locations is used. (Future microcomputers based on the MCS-51™ architecture may incorporate more than 128 bytes of on-chip RAM. Even if this is the case, only the low-order 128 bytes will be directly addressable. The remainder would be accessed indirectly or via the stack pointer.)

Example 3—Adding RAM Location Contents

```
DIRADR ADD CONTENTS OF RAM LOCATION 41H
      TO CONTENTS OF RAM LOCATION 40H
DIRADR MOV A,40H
      ADD A,41H
      MOV 40H,A
```

All I/O ports and special function, control, or status registers are assigned addresses between 128 and 255 (80H-0FFH). When the direct address byte is between these limits the corresponding hardware register is accessed. For example, Ports 0 and 1 are assigned direct addresses 80H and 90H, respectively. A complete list is presented in Table 5. Don't waste your time trying to memorize the addresses in Table 5. Since programs using absolute addresses for function registers would be difficult to write or understand, ASM51 allows and understands the abbreviations listed instead.

Example 4—Adding Input Port Data to Output Port Data

```
PRTADR ADD DATA INPUT ON PORT 1
      TO DATA PREVIOUSLY OUTPUT
      ON PORT 0
PRTADR MOV A,PO
      ADD A,P1
      MOV PO,A
```

Direct addressing allows all special-function registers in the 8051 to be read, written, or used as instruction operands. In general, this is the *only* method used for accessing I/O ports and special-function registers. If direct addressing is used with special-function register addresses other than those listed, the result of the instruction is undefined.

The 8048 does not have or need any generalized direct addressing mode, since there are only five special registers (BUS, P1, P2, PSW, & T) rather than twenty. Instead, 16 special 8048 opcodes control output bits or read or write each register to the accumulator. These functions are all subsumed by four of the 27 direct addressing instructions of the 8051.

Table 5. 8051 Hardware Register Direct Addresses

Register	Address	Function
P0	80H*	Port 0
SP	81H	Stack Pointer
DPL	82H	Data Pointer (Low)
DPH	83H	Data Pointer (High)
TCON	88H*	Timer register
TMOD	89H	Timer Mode register
TL0	8AH	Timer 0 Low byte
TL1	8BH	Timer 1 Low byte
TH0	8CH	Timer 0 High byte
TH1	8DH	Timer 1 High byte
P1	90H*	Port 1
SCON	98H*	Serial Port Control register
SBUF	99H	Serial Port data Buffer
P2	0A0H*	Port 2
IE	0A8H*	Interrupt Enable register
P3	0B0H*	Port 3
IP	0B8H*	Interrupt Priority register
PSW	0D0H*	Program Status Word
ACC	0E0H*	Accumulator (direct address)
B	0F0H*	B register

* = bit addressable register.

Register-Indirect Addressing

How can you handle variables whose locations in RAM are determined, computed, or modified while the program is running? This situation arises when manipulating sequential memory locations, indexed entries within tables in RAM, and multiple precision or string operations. Register or Direct addressing cannot be used, since their operand addresses are fixed at assembly time.

The 8051 solution is "register-indirect RAM addressing." R0 and R1 of each register bank may operate as index or pointer registers, their contents indicating an address into RAM. The internal RAM location so addressed is the actual operand used. The least significant bit of the instruction opcode determines which register is used as the "pointer" (Figure 12.c).

In the 8051 assembly language, register-indirect addressing is represented by a commercial "at" sign ("@") preceding R0, R1, or a symbol defined by the user to be equal to R0 or R1.

Example 5—Indirect Addressing

```

INDADR ADD CONTENTS OF MEMORY LOCATION
        ADDRESSED BY REGISTER 1
        TO CONTENTS OF RAM LOCATION
        ADDRESSED BY REGISTER 0
INDADR MOV A, @R0
ADD A, @R1
MOV @R0, A

```

Indirect addressing on the 8051 is the same as in the 8048 family, except that all eight bits of the pointer register contents are significant; if the contents point to a non-existent memory location (i.e., an address greater than 7FH on the 8051) the result of the instruction is undefined. (Future microcomputers based on the MCS-51™ architecture could implement additional memory in the on-chip RAM logical address space at locations above 7FH.) The 8051 uses register-indirect addressing for five new instructions plus the 13 on the 8048.

Immediate Addressing

When a source operand is a constant rather than a variable (i.e., the instruction uses a value known at assembly time), then the constant can be incorporated into the instruction. An additional instruction byte specifies the value used (Figure 12.d).

The value used is fixed at the time of ROM manufacture or EPROM programming and may not be altered during program execution. In the assembly language immediate operands are preceded by a number sign ("#"). The operand may be either a numeric string, a symbolic variable, or an arithmetic expression using constants.

Example 6—Adding Constants Using Immediate Addressing

```

IMHADR ADD THE CONSTANT 12 (DECIMAL)
        TO THE CONSTANT 34 (DECIMAL)
        LEAVE SUM IN ACCUMULATOR
IMHADR MOV A, #12
ADD A, #34

```

The preceding example was included for consistency; it has little practical value. Instead, ASM51 could compute the sum of two constants at assembly time.

Example 7—Adding Constants Using ASM51 Capabilities

```

ASMSUM LOAD ACC WITH THE SUM OF
        THE CONSTANT 12 (DECIMAL) AND
        THE CONSTANT 34 (DECIMAL)
ASMSUM MOV A, # (12+34)

```

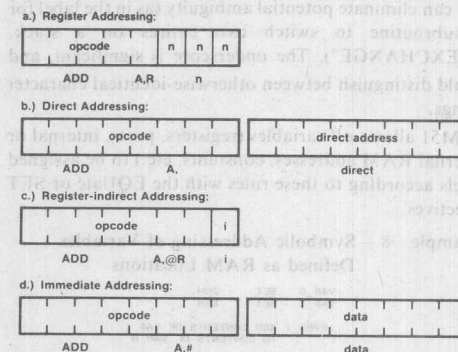


Figure 12. Data Addressing Machine Code Formats

AFN-01502A-19

Addressing Mode Combinations

The above examples all demonstrated the use of the four data-addressing modes in two-operand instructions (MOV, ADD) which use the accumulator as one operand. The operations ADDC, SUBB, ANL, ORL, and XRL (all to be discussed later) could be substituted for ADD in each example. The first three modes may be also be used for the XCH operation or, in combination with the Immediate Addressing mode (and an additional byte), loaded with a constant. The one-operand instructions INC and DEC, DJNZ, and CJNE may all operate on the accumulator, or may specify the Register, Direct, and Register-indirect addressing modes. Exception: as in the 8048, DJNZ cannot use the accumulator or indirect addressing. (The PUSH and POP operations cannot inherently address the accumulator as a special register either. However, all three can *directly* address the accumulator as one of the twenty special-function registers by putting the symbol "ACC" in the operand field.)

Advantages of Symbolic Addressing

Like most assembly or higher-level programming languages, ASM51 allows instructions or variables to be given appropriate, user-defined symbolic names. This is done for instruction lines by putting a label followed by a colon (":") before the instruction proper, as in the above examples. Such symbols must start with an alphabetic character (remember what distinguished BACH from 0BACH?), and may include any combination of letters, numbers, question marks ("?",) and underscores ("_"). For very long names only the first 31 characters are relevant.

Assembly language programs may intermix upper- and lower-case letters arbitrarily, but ASM51 converts both to upper-case. For example, ASM51 will internally process an "I" for an "i" and, of course, "A_TOOTH" for "a_tooth."

The underscore character makes symbols easier to read and can eliminate potential ambiguity (as in the label for a subroutine to switch two entires on a stack, "S_EXCHANGE"). The underscore is significant, and would distinguish between otherwise-identical character strings.

ASM51 allows *all* variables (registers, ports, internal or external RAM addresses, constants, etc.) to be assigned labels according to these rules with the EQUate or SET directives.

Example 8 — Symbolic Addressing of Variables Defined as RAM Locations

```

VAR_0 SET 20H
VAR_1 SET 21H
SYMB_1 ADD CONTENTS OF VAR_1
        TO CONTENTS OF VAR_0
SYMB_1 MOV A, VAR_0
        ADD A, VAR_1
        MOV VAR_0, A

```

Notice from Table 4 that the MCS-51™ instruction set has relatively few instruction mnemonics (abbreviations) for the programmer to memorize. Different data types or addressing modes are determined by the operands specified, rather than variations on the mnemonic. For example, the mnemonic "MOV" is used by 18 different instructions to operate on three data types (bit, byte, and address). The fifteen versions which move byte variables between the logical address spaces are diagrammed in Figure 13. Each arrow shows the direction of transfer from source to destination.

Notice also that for most instructions allowing register addressing there is a corresponding direct addressing instruction and vice versa. This lets the programmer begin writing 8051 programs as if (s)he has access to 128 different registers. When the program has evolved to the point where the programmer has a fairly accurate idea how often each variable is used, he/she may allocate the working registers in each bank to the most "popular" variables. (The assembly cross-reference option will show exactly how often and where each symbol is referenced.) If symbolic addressing is used in writing the source program only the lines containing the symbol definition will need to be changed; the assembler will produce the appropriate instructions even though the rest of the program is left untouched. Editing only the first two lines of Example 8 will shrink the six-byte code segment produced in half.

How are instruction sets "counted"? There is no standard practice; different people assessing the same CPU using different conventions may arrive at different totals.

Each operation is then broken down according to the different addressing modes (or combinations of addressing modes) it can accommodate. The "CLR" mnemonic is used by two instructions with respect to bit variables ("CLR C" and "CLR bit") and once ("CLR A") with regards to bytes. This expansion yields the 111 separate instructions of Table 4.

The method used for the MCS-51 instruction set first breaks it down into "operations": a basic function applied to a single data type. For example, the four versions of the ADD instruction are grouped to form one operation — addition of eight-bit variables. The six forms of the ANL instruction for *byte* variables make up a different operation; the two forms of ANL which operate on *bits* are considered still another. The MOV mnemonic is used by three different operation classes, depending on whether bit, byte, or 16-bit values are affected. Using this terminology the 8051 can perform 51 different operations.

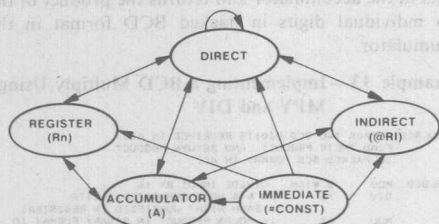


Figure 13. Road map for moving data bytes

Example 9—Redeclaring Example 8 Symbols as Registers

```

VAR_0 SET RO
VAR_1 SET R1
SYMB_2 ADD CONTENTS OF VAR_1
TO CONTENTS OF VAR_0
SYMB_2 MOV A, VAR_0
ADD A, VAR_1
MOV VAR_0, A
  
```

Arithmetic Instruction Usage — ADD, ADDC, SUBB and DA

The ADD instruction adds a byte variable with the accumulator, leaving the result in the accumulator. The carry flag is set if there is an overflow from bit 7 and cleared otherwise. The AC flag is set to the carry-out from bit 3 for use by the DA instruction described later. ADDC adds the previous contents of the carry flag with the two byte variables, but otherwise is the same as ADD.

The SUBB (subtract with borrow) instruction subtracts the byte variable indicated and the contents of the carry flag together from the accumulator, and puts the result back in the accumulator. The carry flag serves as a "Borrow Required" flag during subtraction operations; when a greater value is subtracted from a lesser value (as in subtracting 5 from 1) requiring a borrow into the highest order bit, the carry flag is set; otherwise it is cleared.

When performing signed binary arithmetic, certain combinations of input variables can produce results which seem to violate the Laws of Mathematics. For example, adding 7FH (127) to itself produces a sum of 0FEH, which is the two's complement representation of -2 (refer back to Table 2)! In "normal" arithmetic, two positive values can't have a negative sum. Similarly, it is normally impossible to subtract a positive value from a negative value and leave a positive result — but in two's complement there are instances where this too may happen. Fundamentally, such anomalies occur when the magnitude of the resulting value is too great to "fit" into the seven bits allowed for it; there is no one-byte two's complement representation for 254, the true sum of 127 and 127.

The MCS-51™ processors detect whether these situations occur and indicate such errors with the OV flag. (OV may be tested with the conditional jump instructions JB and JNB, described under the Boolean Processor chapter.)

At a hardware level, OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6. When adding signed integers this indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands; on SUBB this indicates a negative result after subtracting a negative number from a positive number, or a positive result when a positive number is subtracted from a negative number.

The ADDC and SUBB instructions incorporate the previous state of the carry (borrow) flag to allow multiple precision calculations by repeating the operation with successively higher-order operand bytes. In either case, the carry must be cleared before the first iteration.

If the input data for a multiple precision operation is an unsigned string of integers, upon completion the carry flag will be set if an overflow (for ADDC) or underflow (for SUBB) occurs. With two's complement signed data (i.e., if the most significant bit of the original input data indicates the sign of the string), the overflow flag will be set if overflow or underflow occurred.

Example 10—String Subtraction with Signed Overflow Detection

```

SUBSTR SUBTRACT STRING INDICATED BY R1
FROM STRING INDICATED BY R0 TO
PRECISION INDICATED BY R2
CHECK FOR SIGNED OVERFLOW WHEN DONE

SUBSTR CLR C, BORROW=0
SUBS1 MOV A, R0, SUBTRACT NEXT PLACE
SUBB A, R1, BRO, A
MOV R0, A, BUMP POINTERS
INC R1
DJNZ R2, SUBS1, LOOP AS NEEDED
WHEN DONE, TEST IF OVERFLOW OCCURED
ON LAST ITERATION OF LOOP
JNB OV, OV_OK, (OVERFLOW RECOVERY ROUTINE)
OV_OK RET, RETURN
  
```

Decimal addition is possible by using the DA instruction in conjunction with ADD and/or ADDC. The eight-bit binary value in the accumulator resulting from an earlier addition of two variables (each a packed BCD digit-pair) is adjusted to form two BCD digits of four bits each. If the contents of accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag had been set, six is added to the accumulator producing the proper BCD digit in the low-order nibble. (This addition might itself set — but would not clear — the carry flag.) If the carry flag is set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these bits are incremented by six. The carry flag is left set if originally set or if either addition of six produces a carry out of the highest-order bit, indicating the sum of the original two BCD variables is greater than or equal to decimal 100.

Example 11—Two Byte Decimal Add with Registers and Constants

BCDADD ADD THE CONSTANT 1,234 (DECIMAL) TO THE CONTENTS OF REGISTER PAIR (R3:R2) (ALREADY A 4 BCD-DIGIT VARIABLE)

```
BCDADD MOV A, R2
ADD A, #34H
DA A
MOV R2, A
MOV A, R3
ADDC A, #12H
DA A
MOV R3, A
RET
```

Multiplication and Division

The instruction "MUL AB" multiplies the unsigned eight-bit integer values held in the accumulator and B-register. The low-order byte of the sixteen-bit product is left in the accumulator, the higher-order byte in B. If the high-order eight-bits of the product are all zero the overflow flag is cleared; otherwise it is set. The programmer can poll OV to determine when the B register is non-zero and must be processed.

"DIV AB" divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in the B-register. The integer part of the quotient is returned in the accumulator; the remainder in the B-register. If the B-register originally contained 00H then the overflow flag will be set to indicate a division error, and the values returned will be undefined. Otherwise OV is cleared.

The divide instruction is also useful for purposes such as radix conversion or separating bit fields of the accumulator. A short subroutine can convert an eight-bit unsigned binary integer in the accumulator (between 0 & 255) to a three-digit (two byte) BCD representation. The hundred's digit is returned in one register (HUND) and the ten's and one's digits returned as packed BCD in another (TENONE).

Example 12—Use of DIV Instruction for Radix Conversion

BINBCD CONVERT 8-BIT BINARY VARIABLE IN ACC TO 3-DIGIT PACKED BCD FORMAT
HUNDREDS' PLACE LEFT IN VARIABLE 'HUND',
TENS' AND ONES' PLACES IN 'TENONE'

```
HUND EQU 21H
TENONE EQU 22H
```

```
BINBCD MOV B, #100 ; DIVIDE BY 100 TO DETERMINE NUMBER OF HUNDREDS
DIV AB
MOV HUND, A
MOV A, #10 ; DIVIDE REMAINDER BY 10 TO DETERMINE # OF TENS LEFT
XCH A, B
DIV AB ; TENS DIGIT IN ACC, REMAINDER IS ONES DIGIT
SWAP A
ADD A, B ; PACK BCD DIGITS IN ACC
MOV TENONE, A
RET
```

The divide instruction can also separate eight bits of data in the accumulator into sub-fields. For example, packed BCD data may be separated into two nibbles by dividing the data by 16, leaving the high-nibble in the accumulator and the low-order nibble (remainder) in B. The two digits may then be operated on individually or in conjunction with each other. This example receives two packed BCD

digits in the accumulator and returns the product of the two individual digits in packed BCD format in the accumulator.

Example 13—Implementing a BCD Multiply Using MPY and DIV

MULBCD UNPACK TWO BCD DIGITS RECEIVED IN ACC, FIND THEIR PRODUCT, AND RETURN PRODUCT IN PACKED BCD FORMAT IN ACC

```
MULBCD MOV B, #10H ; DIVIDE INPUT BY 16
DIV AB ; A & B HOLD SEPARATED DIGITS (EACH RIGHT JUSTIFIED IN REGISTER)
MUL AB ; A HOLDS PRODUCT IN BINARY FORMAT (0 - 99(DECIMAL) = 0 - 63H)
MOV B, #10 ; DIVIDE PRODUCT BY 10
DIV AB ; A HOLDS # OF TENS, B HOLDS REMAINDER
SWAP A
ORL A, B ; PACK DIGITS
RET
```

Logical Byte Operations — ANL, ORL, XRL

The instructions ANL, ORL, and XRL perform the logical functions AND, OR, and Exclusive-OR on the two byte variables indicated, leaving the results in the first. No flags are affected. (A word to the wise — do not vocalize the first two mnemonics in mixed company.)

These operations may use all the same addressing modes as the arithmetics (ADD, etc.) but unlike the arithmetics, they are not restricted to operating on the accumulator. Directly addressed bytes may be used as the destination with either the accumulator or a constant as the source. These instructions are useful for clearing (ANL), setting (ORL), or complementing (XRL) one or more bits in a RAM, output ports, or control registers. The pattern of bits to be affected is indicated by a suitable mask byte. Use immediate addressing when the pattern to be affected is known at assembly time (Figure 14); use the accumulator versions when the pattern is computed at run-time.

I/O ports are often used for parallel data in formats other than simple eight-bit bytes. For example, the low-order five bits of port 1 may output an alphabetic character code (hopefully) without disturbing bits 7-5. This can be a simple two-step process. First, clear the low-order five pins with an ANL instruction; then set those pins corresponding to ones in the accumulator. (This example assumes the three high-order bits of the accumulator are originally zero.)

Example 14—Reconfiguring Port Size with Logical Byte Instructions

```
OUT_PX: ANL P1, #11100000B ; CLEAR BITS P1.4 - P1.0
ORL P1, A ; SET P1 PINS CORRESPONDING TO SET ACC BITS
RET
```

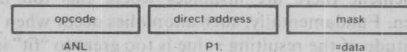


Figure 14. Instruction Pattern for Logical Operation Special Addressing Modes

In this example, low-order bits remaining high may "glitch" low for one machine cycle. If this is undesirable, use a slightly different approach. First, set all pins corresponding to accumulator one bits, then clear the pins corresponding to zeroes in low-order accumulator bits. Not all bits will change from original to final state at the same instant, but no bit makes an intermediate transition.

Example 15—Reconfiguring I/O Port Size without Glitching

```

ALT_PX ORL P1, A
        ANL P1, A
        RET

```

Program Control — Jumps, Calls, Returns

Whereas the 8048 only has a single form of the simple jump instruction, the 8051 has three. Each causes the program to unconditionally jump to some other address. They differ in how the machine code represents the destination address.

LJMP (Long Jump) encodes a sixteen-bit address in the second and third instruction bytes (Figure 15.a); the destination may be anywhere in the 64 Kilobyte program memory address space.

The two-byte AJMP (Absolute Jump) instruction encodes its destination using the same format as the 8048: address bits 10 through 8 form a three bit field in the opcode and address bits 7 through 0 form the second byte (Figure 15.b). Address bits 15-12 are unchanged from the (incremented) contents of the P.C., so AJMP can only be used when the destination is known to be within the same 2K memory block. (Otherwise ASM51 will point out the error.)

A different two-byte jump instruction is legal with any nearby destination, regardless of memory block boundaries or "pages." SJMP (Short Jump) encodes the destination with a program counter-relative address in the second byte (Figure 15.c). The CPU calculates the

destination at run-time by adding the signed eight-bit displacement value to the incremented P.C. Negative offset values will cause jumps up to 128 bytes backwards; positive values up to 127 bytes forwards. (SJMP with 00H in the machine code offset byte will proceed with the following instruction).

In keeping with the 8051 assembly language goal of minimizing the number of instruction mnemonics, there is a "generic" form of the three jump instructions. ASM51 recognizes the mnemonic JMP as a "pseudo-instruction," translating it into the machine instructions LJMP, AJMP, or SJMP, depending on the destination address.

Like SJMP, all conditional jump instructions use relative addressing. JZ (Jump if Zero) and JNZ (Jump if Not Zero) monitor the state of the accumulator as implied by their names, while JC (Jump on Carry) and JNC (Jump on No Carry) test whether or not the carry flag is set. All four are two-byte instructions, with the same format as Figure 15.c. JB (Jump on Bit), JNB (Jump on No Bit) and JBC (Jump on Bit then Clear Bit) can test any status bit or input pin with a three byte instruction; the second byte specifies which bit to test and the third gives the relative offset value.

There are two subroutine-call instructions, LCALL (Long Call) and ACALL (Absolute Call). Each increments the P.C. to the first byte of the following instruction, then pushes it onto the stack (low byte first). Saving both bytes increments the stack pointer by two. The subroutine's starting address is encoded in the same ways as LJMP and AJMP. The generic form of the call operation is the mnemonic CALL, which ASM51 will translate into LCALL or ACALL as appropriate.

The return instruction RET pops the high- and low-order bytes of the program counter successively from the stack, decrementing the stack pointer by two. Program execution continues at the address previously pushed: the first byte of the instruction immediately following the call.

When an interrupt request is recognized by the 8051 hardware, two things happen. Program control is automatically "vectored" to one of the interrupt service routine starting addresses by, in effect, forcing the CPU to process an LCALL instead of the next instruction. This automatically stores the return address on the stack. (Unlike the 8048, no status information is automatically saved.)

Secondly, the interrupt logic is disabled from accepting any other interrupts from the same or lower priority. After completing the interrupt service routine, executing an RETI (Return from Interrupt) instruction will return execution to the point where the background program was interrupted — just like RET — while restoring the interrupt logic to its previous state.

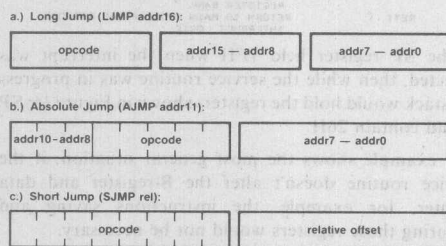


Figure 15. Jump Instruction Machine Code Formats

Operate-and-branch instructions — CJNE, DJNZ

Two groups of instructions combine a byte operation with a conditional jump based on the results.

CJNE (Compare and Jump if Not Equal) compares two byte operands and executes a jump if they disagree. The carry flag is set following the rules for subtraction: if the unsigned integer value of the first operand is less than that of the second it is set; otherwise, it is cleared. However, neither operand is modified.

The CJNE instruction provides, in effect, a one-instruction "case" statement. This instruction may be executed repeatedly, comparing the code variable to a list of "special case" value: the code segment following the instruction (up to the destination label) will be executed only if the operands match. Comparing the accumulator or a register to a series of constants is a convenient way to check for special handling or error conditions; if none of the cases match the program will continue with "normal" processing.

A typical example might be a word processing device which receives ASCII characters through the serial port and drives a thermal hard-copy printer. A standard routine translates "printing" characters to bit patterns, but control characters (, <CR>, <LF>, <BEL>, <ESC> or <SP>) must invoke corresponding special routines. Any other character with an ASCII code less than 20H should be translated into the <NUL> value, 00H, and processed with the printing characters.

Example 16—Case Statements Using CJNE

```
CHAR EQU R7, CHARACTER CODE VARIABLE
INTP CJNE CHAR, #7FH, INTP_1
      (SPECIAL ROUTINE FOR RUBOUT CODE)
INTP_1 RET
      (SPECIAL ROUTINE FOR BELL CODE)
INTP_2 CJNE CHAR, #0AH, INTP_3
      (SPECIAL ROUTINE FOR LFEE CODE)
INTP_3 RET
      (SPECIAL ROUTINE FOR RETURN CODE)
INTP_4 CJNE CHAR, #0DH, INTP_5
      (SPECIAL ROUTINE FOR ESCAPE CODE)
INTP_5 RET
      (SPECIAL ROUTINE FOR SPACE CODE)
INTP_6 JC PRINTC, JUMP IF CODE > 20H
      MOV CHAR, #0, REPLACE CONTROL CHARACTERS WITH
      NULL CODE
PRINTC PROCESS STANDARD PRINTING
      CHARACTER
RET
```

DJNZ (Decrement and Jump if Not Zero) decrements the register or direct address indicated and jumps if the result is not zero, without affecting any flags. This provides a simple means for executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. For example, a 99-usec. software delay loop can be added to code forcing an I/O pin low with only two instructions.

Example 17—Inserting a Software Delay with DJNZ

```
CLR R2, 99
DJNZ R2, $
SETB WR
```

The dollar sign in this example is a special character meaning "the address of this instruction." It is useful in eliminating instruction labels on the same or adjacent source lines. CJNE and DJNZ (like all conditional jumps) use program-counter relative addressing for the destination address.

Stack Operations — PUSH, POP

The PUSH instruction increments the stack pointer by one, then transfers the contents of the single byte variable indicated (direct addressing only) into the internal RAM location addressed by the stack pointer. Conversely, POP copies the contents of the internal RAM location addressed by the stack pointer to the byte variable indicated, then decrements the stack pointer by one.

(Stack Addressing follows the same rules, and addresses the same locations as Register-indirect. Future microcomputers based on the MCS-51™ CPU could have up to 256 bytes of RAM for the stack.)

Interrupt service routines must not change any variable or hardware registers modified by the main program, or else the program may not resume correctly. (Such a change might look like a spontaneous random error.) Resources used or altered by the service routine (Accumulator, PSW, etc.) must be saved and restored to their previous value before returning from the service routine. PUSH and POP provide an efficient and convenient way to save register states on the stack.

Example 18—Use of the Stack for Status Saving on Interrupts

```
LOC_TMP EQU, REMEMBER LOCATION COUNTER
ORG 0003H, STARTING ADDRESS FOR INTERRUPT ROUTINE
LJMP SERVER, JUMP TO ACTUAL SERVICE ROUTINE LOCATED
      ELSEWHERE
ORG LOC_TMP, RESTORE LOCATION COUNTER
PUSH PSW, SAVE ACCUMULATOR (NOTE DIRECT ADDRESSING
      NOTATION)
PUSH B, SAVE B REGISTER
PUSH DPL, SAVE DATA POINTER
PUSH DPH,
MOV PSW, #00001000B, SELECT REGISTER BANK 1
POP DPH, RESTORE REGISTERS IN REVERSE ORDER
POP DPL,
POP B,
POP ACC, RESTORE PSW AND RE-SELECT ORIGINAL
POP PSW, REGISTER BANK
RETI, RETURN TO MAIN PROGRAM AND RESTORE
      INTERRUPT LOGIC
```

If the SP register held 1FH when the interrupt was detected, then while the service routine was in progress the stack would hold the registers shown in Figure 16: SP would contain 26H.

The example shows the most general situation; if the service routine doesn't alter the B-register and data pointer, for example, the instructions saving and restoring those registers would not be necessary.

The stack may also pass parameters to and from subroutines. The subroutine can indirectly address the parameters derived from the contents of the stack pointer.

RAM ADDR	
7FH	
26H	DPH ← (SP)
25H	DPL
24H	B
23H	ACC
22H	PSW
21H	PC (HIGH)
20H	PC (LOW)
1FH	
00H	

Figure 16. Stack contents during interrupt

One advantage here is simplicity. Variables need not be allocated for specific parameters, a potentially large number of parameters may be passed, and different calling programs may use different techniques for determining or handling the variables.

For example, the following subroutine reads out a parameter stored on the stack by the calling program, uses the low order bits to access a local look-up table holding bit patterns for driving the coils of a four phase stepper motor, and stores the appropriate bit pattern back in the same position on the stack before returning. The accumulator contents are left unchanged.

Example 19—Passing Variable Parameters to Subroutines Using the Stack

```

NEXTPOS MOV RO, SP      ; ACCESS LOCATION PARAMETER PUSHED INTO
DEC RO      ;
DEC RO      ;
XCH A, @RO   ; READ INPUT PARAMETER AND SAVE IN ACCUMULATOR
ANL A, #03H  ; MASK ALL BUT LOW-ORDER TWO BITS
ADD A, #2    ; ALLOW FOR OFFSET FROM MOVX TO TABLE
MOVC A, @A+PC ; READ LOOK-UP TABLE ENTRY
XCH A, @RO   ; PASS BACK TRANSLATED VALUE AND RESTORE ACC
RET          ; RETURN TO BACKGROUND PROGRAM
STPTBL DB 01011111B ; POSITION 0
DB 01011111B ; POSITION 1
DB 10011111B ; POSITION 2
DB 10101111B ; POSITION 3

```

The background program may reach this subroutine with several different calling sequences, all of which PUSH a value before calling the routine and POP the result after. A motor on Port 1 may be initialized by placing the desired position (zero) on the stack before calling the subroutine and outputting the results directly to a port afterwards.

Example 20—Sending and Receiving Data Parameters Via the Stack

```

CLR A
PUSH ACC
CALL NEXTPOS
POP P1

```

If the position of the motor is determined by the contents of variable POSM1 (a byte in internal RAM) and the position of a second motor on Port 2 is determined by the data input to the low-order nibble of Port 2, a six-instruction sequence could update them both.

Example 21—Loading and Unloading Stack Direct from I/O Ports

```

POSM1 EQU 51
PUSH POSM1
CALL NEXTPOS
POP P1
PUSH P2
CALL NXTPOS
POP P2

```

Data Pointer and Table Look-up instructions — MOV, INC, MOVC, JMP

The data pointer can be loaded with a 16-bit value using the instruction MOV DPTR, #data16. The data used is stored in the second and third instruction bytes, high-order byte first. The data pointer is incremented by INC DPTR. A 16-bit increment is performed; an overflow from the low byte will carry into the high-order byte. Neither instruction affects any flags.

The MOVC (Move Constant) instructions (MOVC A, @A+DPTR and MOVC A, @A+PC) read into the accumulator bytes of data from the program memory logical address space. Both use a form of indexed addressing: the former adds the unsigned eight-bit accumulator contents with the sixteen-bit data pointer register, and uses the resulting sum as the address from which the byte is fetched. A sixteen-bit addition is performed; a carry-out from the low-order eight bits may propagate through higher-order bits, but the contents of the DPTR are not altered. The latter form uses the incremented program counter as the "base" value instead of the DPTR (figure 17). Again, neither version affects the flags.

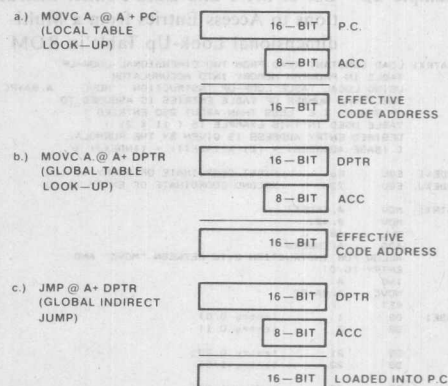


Figure 17. Operation of MOVC instructions

AFN-01502A-25

Each can be part of a three step sequence to access look-up tables in ROM. To use the DPTR-relative version, load the Data Pointer with the starting address of a look-up table; load the accumulator with (or compute) the index of the entry desired; and execute `MOVC A,@A+DPTR`. Unlike the similar `MOVP3` instructions in the 8048, the table may be located anywhere in program memory. The data pointer may be loaded with a constant for short tables. Or to allow more complicated data structures, or tables with more than 256 entries, the values for DPH and DPL may be computed or modified with the standard arithmetic instruction set.

The PC-relative version has the advantage of not affecting the data pointer. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction to the start of the table by adding the number of bytes separating them to the accumulator; then execute the `MOVC A,@A+PC` instruction.

Let's look at a non-trivial situation where this instruction would be used. Some applications store large multi-dimensional look-up tables of dot matrix patterns, non-linear calibration parameters, and so on in a linear (one-dimensional) vector in program memory. To retrieve data from the tables, variables representing matrix indices must be converted to the desired entry's memory address. For a matrix of dimensions (MDIMEN x NDIMEN) starting at address BASE and respective indices INDEXI and INDEXJ, the address of element (INDEXI, INDEXJ) is determined by the formula,

$$\text{Entry Address} = \text{BASE} + (\text{NDIMEN} \times \text{INDEXI}) + \text{INDEXJ}$$

The code shown below can access any array with less than 255 entries (i.e., an 11x21 array with 231 elements). The table entries are defined using the Data Byte ("DB") directive, and will be contained in the assembly object code as part of the accessing subroutine itself.

Example 22—Use of MPY and Data Pointer Instructions to Access Entries from a Multi-dimensional Look-Up Table in ROM

```

; MATRIX1 LOAD CONSTANT READ FROM TWO DIMENSIONAL LOOK-UP
; TABLE IN PROGRAM MEMORY INTO ACCUMULATOR
; USING LOCAL TABLE LOOK-UP INSTRUCTION, MOVC A,@A+PC
; THE TOTAL NUMBER OF TABLE ENTRIES IS ASSUMED TO
; BE SMALL, I.E. LESS THAN ABOUT 250 ENTRIES.
; TABLE USED IN THIS EXAMPLE IS ( 11 X 21 )
; DESIRED ENTRY ADDRESS IS GIVEN BY THE FORMULA,
; ( (BASE ADDRESS) + (21 X INDEXI) ) + (INDEXJ)
;
INDEXI EQU R6 ; FIRST COORDINATE OF ENTRY (0-10)
INDEXJ EQU R7 ; SECOND COORDINATE OF ENTRY (0-20)
;
MATRIX1 MOV A,INDEXI
MOV B,#21
MUL AB
ADD A,INDEXJ
; ALLOW FOR INSTRUCTION-BYTE BETWEEN "MOVC" AND
; ENTRY (0,0)
INC A
MOVC A,@A+PC
RET
;
BASE1 DB 1 ; (entry 0,0)
DB 2 ; (entry 0,1)
;
DB 21 ; (entry 0,20)
DB 22 ; (entry 1,0)
;
DB 42 ; (entry 1,20)
;
DB 231 ; (entry 10,20)

```

There are several different means for branching to sections of code determined or selected at run time. (The single destination addresses incorporated into conditional and unconditional jumps are, of course, determined at assembly time). Each has advantages for different applications.

The most common is an N-way conditional jump based on some variable, with all of the potential destinations known at assembly time. One of a number of small routines is selected according to the value of an index variable determined while the program is running. The most efficient way to solve this problem is with the `MOVC` and an indirect jump instruction, using a short table of one byte offset values in ROM to indicate the relative starting addresses of the several routines.

`JMP @A+DPTR` is an instruction which performs an indirect jump to an address determined during program execution. The instruction adds the eight-bit unsigned accumulator contents with the contents of the sixteen-bit data pointer, just like `MOVC A,@A+DPTR`. The resulting sum is loaded into the program counter and is used as the address for subsequent instruction fetches. Again, a sixteen-bit addition is performed; a carry out from the low-order eight bits may propagate through the higher-order bits. In this case, neither the accumulator contents nor the data pointer is altered.

The example subroutine below reads a byte of RAM into the accumulator from one of four alternate address spaces, as selected by the contents of the variable `MEMSEL`. The address of the byte to be read is determined by the contents of `R0` (and optionally `R1`). It might find use in a printing terminal application, where four different model printers all use the same ROM code but use different types and sizes of buffer memory for different speeds and options.

Example 23—N-Way Branch and Computed Jump Instructions via `JMP @ADPTR`

```

MEMSEL EQU R3
;
JUMP_4 MOV A, MEMSEL
MOV DPTR, #JMP1TBL
MOVC A,@A+DPTR
JMP @A+DPTR
;
JMP1TBL DB MEMSP0-JMP1TBL
DB MEMSP1-JMP1TBL
DB MEMSP2-JMP1TBL
DB MEMSP3-JMP1TBL
;
MEMSP0 MOV RET ; READ FROM INTERNAL RAM
MEMSP1 MOVX RET ; READ FROM 256 BYTES OF EXTERNAL RAM
MEMSP2 MOV DPL,R0
MOV DPH,R1
MOVC A,@DPTR ; READ FROM 64K BYTES OF EXTERNAL RAM
RET
MEMSP3 MOV A,R1
ANL A,#07H
ANL P1,#11111000B
ORL P1,A
MOVX A,@R0 ; READ FROM 4K BYTES OF EXTERNAL RAM
RET

```

Note that this approach is suitable whenever the size of jump table plus the length of the alternate routines is less than 256 bytes. The jump table and routines may be located anywhere in program memory, independent of 256-byte program memory pages.

For applications where up to 128 destinations must be selected, all of which reside in the same 2K page of program memory which may be reached by the two-byte absolute jump instructions, the following technique may be used. In the above mentioned printing terminal example, this sequence could "parse" 128 different codes for ASCII characters arriving via the 8051 serial port.

Example 24—N-Way Branch with 128 Optional Destinations

OPTION	EQU	R3
JMP12B	Mov	A, OPTION
	RL	A
	DPTR, #INSTRBL	FIRST ENTRY IN JUMP TABLE
	JMP	@A+DPTR JUMP INTO JUMP TABLE
INSTBL	AJMP	PROG00 .128 CONSECUTIVE
	AJMP	PROC01 .AJMP INSTRUCTIONS
	AJMP	PROC02
	AJMP	PROC7E
	END	

The destinations in the jump table (PROC00-PROC07F) are not all necessarily unique routines. A large number of special control codes could each be processed with their own unique routine, with the remaining printing characters all causing a branch to a common routine for entering the character into the output queue.

In those rare situations where even 128 options are insufficient, or where the destination routines may cross a 2K page boundary, the above approach may be modified slightly as shown below.

Example 25—256-Way Branch Using Address Look-Up Tables

```

RTEMP EQU R7

JMP256 MOV DPTR,#ADRTBL ; FIRST ENTRY IN TABLE OF ADDRESSES
        MOVB A,OPTION
        CLRB C
        RLCL A          ; MULTIPLY BY 2 FOR 2 BYTE JUMP TABLE
        JNCL LDIW28
        INC DPH
        MOV RTEMP,A      ; SAVE ACC FOR HIGH BYTE READ
LOWI28   MOVC A,&A+DPTR    ; READ LOW BYTE FROM JUMP TABLE
        XCHC A           ;
        INC A            ;
        MLCB A,&A+DPTR    ; GET LOW-ORDER BYTE FROM TABLE
        PUSH ACC
        MOV A,RTEMP
        MOVC A,&A+DPTR    ; GET HIGH-ORDER BYTE FROM TABLE
        PUSH ACC
        ; THE TWO ACC VALUES HAVE PRODUCED
        ; A "RETURN ADDRESS" ON THE STACK WHICH CORRESPONDS
        ; TO THE DESIRED STARTING ADDRESS
        ; IT MAY BE REACHED BY POPPING THE STACK
        ; INTO THE PC
        RET
ADRTBL DW PROCOO ,UP TO 256 CONSECUTIVE DATA
       DW PROC01 ,WORDS INDICATING STARTING ADDRESSES
       .
       .
       .
DUMMY CODE ADDRESS DEFINITIONS NEEDED BY ABOVE
TWO EXAMPLES:

PROCOO NOP
PROC01 NOP
PROC02 NOP
PROC0E NOP
PROC7F NOP
PROCCF NOP
```

4. BOOLEAN PROCESSING INSTRUCTIONS

The commonly accepted terms for tasks at either end of the computational vs. control application spectrum are, respectively, "number-crunching" and "bit-banging".

Prior to the introduction of the MCS-51™ family, nice number-crunchers made bad bit-bangers and vice versa. The 8051 is the industry's first single-chip micro-computer designed to crunch **and** bang. (In some circles, the latter technique is also referred to as "bit-tiddling". Either is correct.)

Direct Bit Addressing

A number of instructions operate on Boolean (one-bit) variables, using a direct bit addressing mode comparable to direct byte addressing. An additional byte appended to the opcode specifies the Boolean variable, I/O pin, or control bit used. The state of any of these bits may be tested for "true" or "false" with the conditional branch instructions JB (Jump on Bit) and JNB (Jump on Not Bit). The JBC (Jump on Bit and Clear) instruction combines a test-for-true with an unconditional clear.

As in direct byte addressing, bit 7 of the address byte switches between two physical address spaces. Values between 0 and 127 (00H-7FH) define bits in internal RAM locations 20H to 2FH (Figure 18a); address bytes between 128 and 255 (80H-0FFH) define bits in the 2 x "special-function" register address space (Figure 18b). If no 2 x "special-function" register corresponds to the direct bit address used, the result of the instruction is undefined.

Bits so addressed have many wondrous properties. They may be set, cleared, or complemented with the two byte instructions SETB, CLR, or CPL. Bits may be moved to and from the carry flag with MOV. The logical ANL and ORL functions may be performed between the carry and either the addressed bit or its complement.

Bit Manipulation Instructions — MOV

The "MOV" mnemonic can be used to load an addressable bit into the carry flag ("MOV C, bit") or to copy the state of the carry to such a bit ("MOV bit, C"). These instructions are often used for implementing serial I/O algorithms via software or to adapt the standard I/O port structure.

It is sometimes desirable to "re-arrange" the order of I/O pins because of considerations in laying out printed circuit boards. When interfacing the 8051 to an immediately adjacent device with "weighted" input pins, such as keyboard column decoder, the corresponding pins are likely to be not aligned (Figure 19).

There is a trade-off in "scrambling" the interconnections with either interwoven circuit board traces or through software. This is extremely cumbersome (if not impossible) to do with byte-oriented computer architectures. The 8051's unique set of Boolean instructions makes it simple to move individual bits between arbitrary locations.

a.) RAM Bit Addresses.

RAM BYTE	(MSB)							(LSB)
7FH								
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00
1FH	Bank 3							
18H	Bank 2							
17H	Bank 2							
10H	Bank 1							
0FH	Bank 1							
08H	Bank 0							
07H	Bank 0							
00H	Bank 0							

b.) Hardware Register Bit Addresses.

Direct Byte Address	(MSB)	Bit Addresses	(LSB)	Hardware Register Symbol
0FFH				
0FH	F7 F6 F5 F4 F3 F2 F1 F0			B
0E0H	E7 E6 E5 E4 E3 E2 E1 E0			ACC
0D0H	D7 D6 D5 D4 D3 D2 D1 D0			PSW
0B8H	— — — BC BB BA B9 B8			IP
0B0H	B7 B6 B5 B4 B3 B2 B1 B0			P3
0A8H	AF — — AC AB AA A9 A8			IE
0A0H	A7 A6 A5 A4 A3 A2 A1 A0			P2
98H	9F 9E 9D 9C 9B 9A 99 98			SCON
90H	97 96 95 94 93 92 91 90			P1
88H	8F 8E 8D 8C 8B 8A 89 88			TCON
80H	87 86 85 84 83 82 81 80			P0

Figure 18. Bit Address Maps

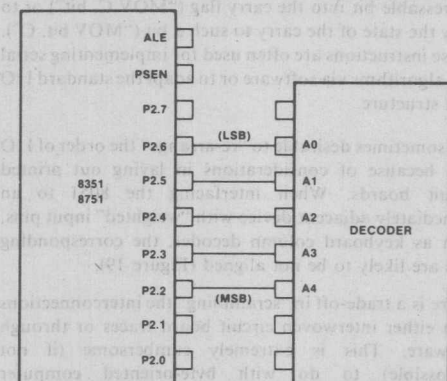


Figure 19. "Mismatch" Between I/O port and Decoder

Example 26 — Re-ordering I/O Port Configuration

```

OUT_P2: RRC A ; MOVE ORIGINAL ACC 0 INTO CY
        MOV P2 6.C ; STORE CARRY TO PIN P26
        RRC A ; MOVE ORIGINAL ACC 1 INTO CY
        MOV P2 5.C ; STORE CARRY TO PIN P25
        RRC A ; MOVE ORIGINAL ACC 2 INTO CY
        MOV P2 4.C ; STORE CARRY TO PIN P24
        RRC A ; MOVE ORIGINAL ACC 3 INTO CY
        MOV P2 3.C ; STORE CARRY TO PIN P23
        RRC A ; MOVE ORIGINAL ACC 4 INTO CY
        MOV P2 2.C ; STORE CARRY TO PIN P22
        RET

```

Solving Combinatorial Logic Equations — ANL, ORL

Virtually all hardware designers are familiar with the problem of solving complex functions using combinatorial logic. The technologies involved may vary greatly, from multiple contact relay logic, vacuum tubes, TTL, or CMOS to more esoteric approaches like fluidics, but in each case the goal is the same; a Boolean (true false) function is computed on a number of Boolean variables.

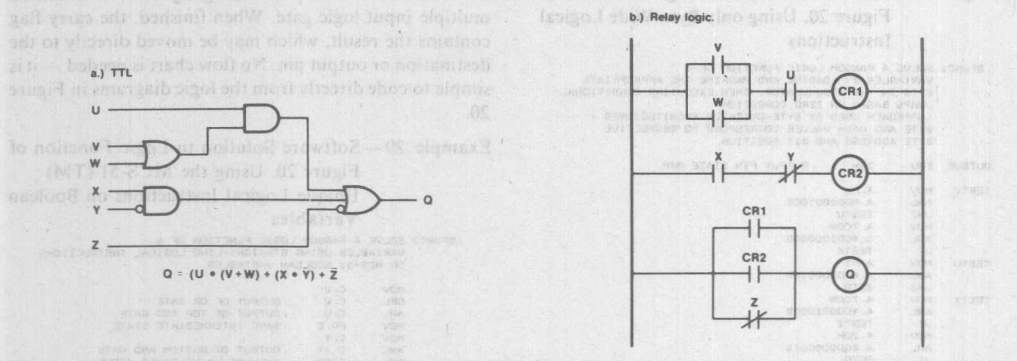


Figure 20. Implementations of Boolean functions

2

Figure 20 shows the logic diagram for an arbitrary function of six variables named U through Z using standard logic and relay logic symbols. Each is a solution of the equation.

$$Q = (U * (V * W)) + (X * Y) * Z$$

(While this equation could be reduced using Karnaugh Maps or algebraic techniques, that is not the purpose of this example. Even a minor change to the function equation would require re-reducing from scratch.)

Most digital computers can solve equations of this type with standard word-wide logical instructions and conditional jumps. Still, such software solutions seem somewhat sloppy because of the many paths through the program the computation can take.

Assume U and V are input pins being read by different input ports, W and X are status bits for two peripheral controllers (read as I/O ports), and Y and Z are software flags set or cleared earlier in the program. The end result must be written to an output pin on some third port.

For the sake of comparison we will implement this function with software drawn from three proper subsets of the MCS-51™ instruction set. The first two implementations follow the flow chart shown in Figure 21. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP. These exits then write the output port with the data previously written to the same port with the result bit respectively one or zero.

In the first case, we assume there are no instructions for addressing individual bits other than special flags like the carry. This is typical of many older microprocessors and mainframe computers designed for number-crunching. MCS-51™ mnemonics are used here, though for most other machines the issue would be even further clouded by their use of operation-specific mnemonics like

INPUT, OUTPUT, LOAD, STORE, etc., instead of the universal MOV.

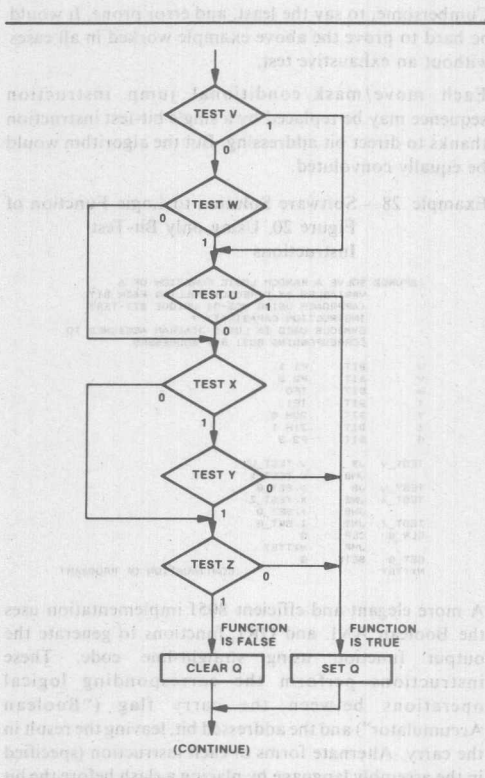


Figure 21. Flow chart for tree-branching logic implementation

Example 27—Software Solution to Logic Function of Figure 20, Using only Byte-Wide Logical Instructions

```

;BFUNC1 SOLVE A RANDOM LOGIC FUNCTION OF 6
;VARIABLES BY LOADING AND MASKING THE APPROPRIATE
;BITS IN THE ACCUMULATOR, THEN EXECUTING CONDITIONAL
;JUMPS BASED ON ZERO CONDITION
;(APPROACH USED BY BYTE-ORIENTED ARCHITECTURES)
;BYTE AND MASK VALUES CORRESPOND TO RESPECTIVE
;BYTE ADDRESS AND BIT POSITION

```

```

OUTBUF EQU 22H ;OUTPUT PIN STATE MAP
TESTV MOV A,P3
ANL A,#00000100B
JNZ TESTU
MOV A,TC0N
ANL A,#00100000B
JZ TESTX
TESTU MOV A,P1
ANL A,#00000010B
JNZ SETQ
TESTX MOV A,TC0N
ANL A,#00001000B
JZ TESTZ
MOV A,20H
ANL A,#00000001B
JZ SETQ
TESTZ MOV A,21H
ANL A,#00000001B
JZ SETQ
CLRQ MOV A,OUTBUF
ANL A,#11110111B
JMP OUTG
SETQ MOV A,OUTBUF
ORL A,#00001000B
MOV OUTG,A
MOV P3,A

```

Cumbersome, to say the least, and error prone. It would be hard to prove the above example worked in all cases without an exhaustive test.

Each move/mask/conditional jump instruction sequence may be replaced by a single bit-test instruction thanks to direct bit addressing. But the algorithm would be equally convoluted.

Example 28—Software Solution to Logic Function of Figure 20, Using only Bit-Test Instructions

```

;BFUNC2 SOLVE A RANDOM LOGIC FUNCTION OF 6
;VARIABLES BY DIRECTLY POLLING EACH BIT
;(APPROACH USING MCS-51 UNIQUE BIT-TEST
;INSTRUCTION CAPABILITY)
;SYMBOLS USED IN LOGIC DIAGRAM ASSIGNED TO
;CORRESPONDING 8051 BIT ADDRESSES

```

```

U BIT P1.1
V BIT P2.2
W BIT TFO
X BIT IE1
Y BIT 20H.0
Z BIT 21H.1
Q BIT P3.3
TEST_V JB V,TEST_U
TEST_U JNB W,TEST_X
TEST_X JNB X,TEST_Z
TEST_Z JNB Y,SET_Q
TEST_Z CLR Z,SET_Q
SET_Q JNB Q,NXTTST
NXTTST SETB Q

```

A more elegant and efficient 8051 implementation uses the Boolean ANL and ORL functions to generate the output function using straight-line code. These instructions perform the corresponding logical operations between the carry flag ("Boolean Accumulator") and the addressed bit, leaving the result in the carry. Alternate forms of each instruction (specified in the assembly language by placing a slash before the bit name) use the complement of the bit's state as the input operand.

These instructions may be "strung together" to simulate a multiple input logic gate. When finished, the carry flag contains the result, which may be moved directly to the destination or output pin. No flow chart is needed — it is simple to code directly from the logic diagrams in Figure 20.

Example 29—Software Solution to Logic Function of Figure 20, Using the MCS-51 (TM) Unique Logical Instructions on Boolean Variables

```

;BFUNC3 SOLVE A RANDOM LOGIC FUNCTION OF 6
;VARIABLES USING STRAIGHT-LINE LOGICAL INSTRUCTIONS
;ON MCS-51 BOOLEAN VARIABLES
MOV C,V ;OUTPUT OF OR GATE
ORL C,W ;OUTPUT OF TOP AND GATE
MOV FO,C ;SAVE INTERMEDIATE STATE
MOV C,X ;OUTPUT OF BOTTOM AND GATE
ANL C,Y ;INCLUDE VALUE SAVED ABOVE
ORL C,F0 ;INCLUDE LAST INPUT VARIABLE
MOV Q,C ;OUTPUT COMPUTED RESULT

```

Simplicity itself. Fast, flexible, reliable, easy to design, and easy to debug.

The Boolean features are useful and unique enough to warrant a complete Application Note of their own. Additional uses and ideas are presented in Application Note AP-70, Using the Intel® MCS-51® Boolean Processing Capabilities, publication number 421519.

5. ON-CHIP PERIPHERAL FUNCTION OPERATION AND INTERFACING

I/O Ports

The I/O port versatility results from the "quasi-bidirectional" output structure depicted in Figure 22. (This is effectively the structure of ports 1, 2, and 3 for normal I/O operations. On port 0 resistor R2 is disabled except during multiplexed bus operations, providing

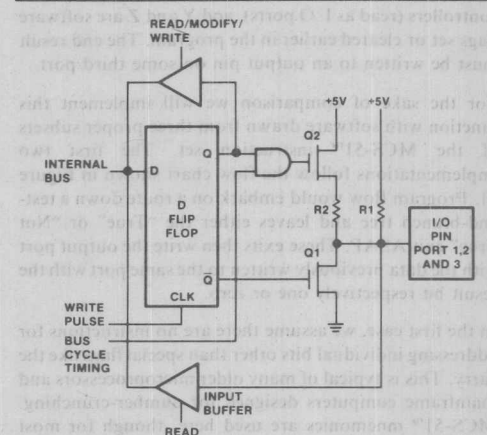


Figure 22. Pseudo-bidirectional I/O port circuitry

AFN-01502A-30

essentially open-collector outputs. For full electrical characteristics see the User's Manual.)

An output latch bit associated with each pin is updated by direct addressing instructions when that port is the destination. The latch state is buffered to the outside world by R1 and Q1, which may drive a standard TTL input. (In TTL terms, Q1 and R1 resemble an open-collector output with a pull-up resistor to Vcc.)

R2 and Q2 represent an "active pull-up" device enabled momentarily when a 0 previously output changes to a 1. This "jerks" the output pin to a 1 level more quickly than the passive pull-up, improving rise-time significantly if the pin is driving a capacitive load. Note that the active pull-up is **only** activated on 0-to-1 transitions at the output latch (unlike the 8048, in which Q2 is activated whenever a 1 is written out).

Operations using an input port or pin as the source operand use the logic level of the pin itself, rather than the output latch contents. This level is affected by both the microcomputer itself and whatever device the pin is connected to externally. The value read is essentially the "OR-tied" function of Q1 and the external device. If the external device is high-impedance, such as a logic gate input or a three state output in the third state, then reading a pin will reflect the logic level previously output. To use a pin for input, the corresponding output latch must be set. The external device may then drive the pin with either a high or low logic signal. Thus the same port may be used as both input and output by writing ones to all pins used as inputs on output operations, and ignoring all pins used as output on an input operation.

In one operand instructions (INC, DEC, DJNZ and the Boolean CPL) the output latch rather than the input pin level is used as the source data. Similarly, two operand instructions using the port as both one source and the destination (ANI, ORI, XRI) use the output latches. This ensures that latch bits corresponding to pins used as inputs will not be cleared in the process of executing these instructions.

The Boolean operation JBC tests the output latch bit, rather than the input pin, in deciding whether or not to jump. Like the byte-wise logical operations, Boolean operations which modify individual pins of a port leave the other bits of the output latch unchanged.

A good example of how these modes may play together may be taken from the host-processor interface expected by an 8243 I/O expander. Even though the 8051 does not include 8048-type instructions for interfacing with an 8243, the parts can be interconnected (Figure 23) and the protocol may be emulated with simple software.

Example 30—Mixing Parallel Output, Input, and Control Strobes on Port 2

```
IN8243 INPUT DATA FROM AN 8243 I/O EXPANDER
CONNECTED TO P23-P20
P25 & P24 MIMIC CS & PROG
P27-P26 USED AS INPUTS
P27 TO BE READ IN AC7
```

```
IN8243 OF A #10100000
MOV P1, A OUTPUT INSTRUCTION CODE
CLR P1, 4 FALLING EDGE OF PROG
OPL P2, #00001111B SET FOR INPUT
MOV A, P2 READ INPUT DATA
SETI 4 RETURN PROG HI-W
SETI 5 DE-SELECT CHIP
```

Serial Port and Timer applications

Configuring the 8051's Serial Port for a given data rate and protocol requires essentially three short sections of software. On power-up or hardware reset the serial port and timer control words must be initialized to the appropriate values. Additional software is also needed in the transmit routine to load the serial port data register and in the receive routine to unload the data as it arrives.

This is best illustrated through an arbitrary example. Assume the 8051 will communicate with a CRT operating at 2400 baud (bits per second). Each character is transmitted as seven data bits, odd parity, and one stop bit. This results in a character rate of 2400 10=240 characters per second.

For the sake of clarity, the transmit and receive subroutines are driven by simple-minded software status polling code rather than interrupts. (It might help to refer back to Figures 7-9 showing the control word formats.) The serial port must be initialized to 8-bit UART mode (M0, M1=01), enabled to receive all messages (M2=0, REN=1). The flag indicating that the transmit register is free for more data will be artificially set in order to let the output software know the output register is available. This can all be set up with one instruction.

Example 31—Serial Port Mode and Control Bits

```
SPINIT INITIALIZE SERIAL PORT
FOR 8-BIT UART MODE
& SET TRANSMIT READY FLAG
```

```
SPINIT MOV SCON, #01010010B
```

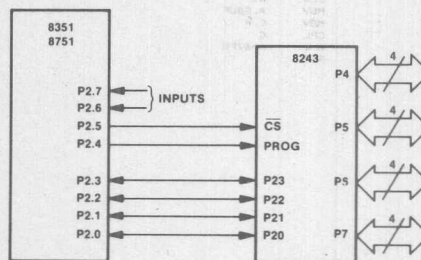


Figure 23. Connecting an 8051 with an 8243 I/O Expander

Timer 1 will be used in auto-reload mode as a data rate generator. To achieve a data rate of 2400 baud, the timer must divide the 1 MHz internal clock by 32 x (desired data rate):

$$\frac{1 \times 10^6}{(32)(2400)}$$

which equals 13.02 rounded down to 13 instruction cycles. The timer must reload the value -13, or 0F3H. (ASM51 will accept both the signed decimal or hexadecimal representations.)

Example 32—Initializing Timer Mode and Control Bits

```

T1INIT: INITIALIZE TIMER 1 FOR
        AUTO-RELOAD AT 32*2400 HZ
        (TO USED AS GATED 16-BIT COUNTER)

T1INIT: MOV     TCON, #11010010B
        MOV     TH1, #-13
        SETB    TR1

```

A simple subroutine to transmit the character passed to it in the accumulator must first compute the parity bit, insert it into the data byte, wait until the transmitter is available, output the character, and return. This is nearly as easy said as done.

Example 33—Code for UART Output, Adding Parity, Transmitter Loading

```

; SP_OUT ADD ODD PARITY TO ACC AND
; TRANSMIT WHEN SERIAL PORT READY

SP_OUT: MOV     C, P
        MOV     ACC, 7; C
        JNB     TI, $
        CLR     TI
        MOV     SBUF, A
        RET

```

A simple minded routine to wait until a character is received, set the carry flag if there is an odd-parity error, and return the masked seven-bit code in the accumulator is equally short.

Example 34—Code for UART Reception and Parity Verification

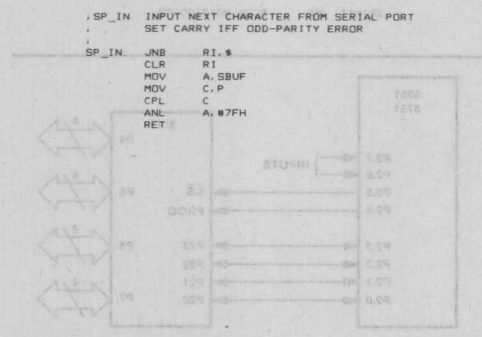


Figure 23. Connecting an 8085 with an 8255 PPI

6. SUMMARY

This Application Note has described the architecture, instruction set, and on-chip peripheral features of the first three members of the MCS-51™ microcomputer family. The examples used throughout were admittedly (and necessarily) very simple. Additional examples and techniques may be found in the MCS-51™ User's Manual and other application notes written for the MCS-48™ and MCS-51™ families.

Since its introduction in 1977, the MCS-48™ family has become the industry standard single-chip microcomputer. The MCS-51™ architecture expands the addressing capabilities and instruction set of its predecessor while ensuring flexibility for the future, and maintaining basic software compatibility with the past.

Designers already familiar with the 8048 or 8049 will be able to take with them the education and experience gained from past designs as ever-increasing system performance demands force them to move on to state-of-the-art products. Newcomers will find the power and regularity of the 8051 instruction set an advantage in streamlining both the learning and design processes.

Microcomputer system designers will appreciate the 8051 as basically a single-chip solution to many problems which previously required board-level computers. Designers of real-time control systems will find the high execution speed, on-chip peripherals, and interrupt capabilities vital in meeting the timing constraints of products previously requiring discrete logic designs. And designers of industrial controllers will be able to convert ladder diagrams directly from tested-and-true TTL or relay-logic designs to microcomputer software, thanks to the unique Boolean processing capabilities.

It has not been the intent of this note to gloss over the difficulty of designing microcomputer-based systems. To be sure, the hardware and software design aspects of any new computer system are nontrivial tasks. However, the system speed and level of integration of the MCS-51™ microcomputers, the power and flexibility of the instruction set, and the sophisticated assembler and other support products combine to give both the hardware and software designer as much of a head start on the problem as possible.

The Boolean operation IBC tests the output of the instruction. The output of the instruction is the output of the instruction. The output of the instruction is the output of the instruction.

A good example of how these modes may be used together may be taken from the host-processor interface expected from an 8255 PPI. Even though the 8085 does not include 8084-type instructions for interfacing with an 8255, the parts can be interfaced (Figure 23) and the control may be completed with simple software.

PAGE	CONTENTS
2-32	1.0 INTRODUCTION
2-34	2.0 BOOLEAN PROCESSOR OPERATION
2-35	Processing Elements
2-37	Direct Bit Addressing
2-40	Instruction Set
2-42	Simple Instruction Combinations
2-44	3.0 BOOLEAN PROCESSOR APPLICATIONS
2-44	Design Example #1—Bit Permutation
2-46	Design Example #2—Software Serial I/O
2-51	Design Example #3—Combinational Logic Equations
2-52	Design Example #4—Automotive Dashboard Functions
2-52	Design Example #5—Complex Control Functions
2-71	Additional Functions and Uses
2-75	4.0 SUMMARY

Using the Intel MCS®-51 Boolean Processing Capabilities

JOHN WHARTON
MICROCONTROLLER APPLICATIONS

April 1980

2

Order Number: 203830-001

USING THE INTEL MCS®-51 BOOLEAN PROCESSING CAPABILITIES

CONTENTS

PAGE

1.0 INTRODUCTION	2-33
2.0 BOOLEAN PROCESSOR OPERATION	2-34
Processing Elements	2-35
Direct Bit Addressing	2-37
Instruction Set	2-40
Simple Instruction Combinations	2-42
3.0 BOOLEAN PROCESSOR APPLICATIONS	2-44
Design Example #1—Bit Permutation ...	2-44
Design Example #2—Software Serial I/O	2-49
Design Example #3—Combinational Logic Equations	2-51
Design Example #4—Automotive Dashboard Functions	2-55
Design Example #5—Complex Control Functions	2-62
Additional Functions and Uses	2-71
4.0 SUMMARY	2-72
APPENDIX A	2-73

1.0 INTRODUCTION

The Intel microcontroller family now has three new members: the Intel® 8031, 8051, and 8751 single-chip microcomputers. These devices, shown in Figure 1, will allow whole new classes of products to benefit from recent advances in Integrated Electronics. Thanks to Intel's new HMOS technology, they provide larger program and data memory spaces, more flexible I/O and peripheral capabilities, greater speed, and lower system cost than any previous-generation single-chip microcomputer.

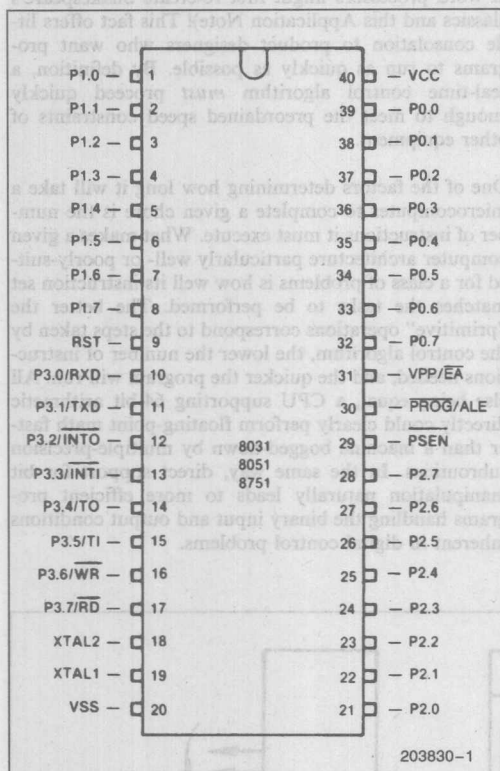


Figure 1. 8051 Family Pinout Diagram

Table 1 summarizes the quantitative differences between the members of the MCS®-48 and 8051 families. The 8751 contains 4K bytes of EPROM program memory fabricated on-chip, while the 8051 replaces the EPROM with 4K bytes of lower-cost mask-programmed ROM. The 8031 has no program memory on-chip; instead, it accesses up to 64K bytes of program memory from external memory. Otherwise, the three new family members are identical. Throughout this Note, the term "8051" will represent all members of the 8051 Family, unless specifically stated otherwise.

The CPU in each microcomputer is one of the industry's fastest and most efficient for numerical calculations on byte operands. But controllers often deal with bits, not bytes: in the real world, switch contacts can only be open or closed, indicators should be either lit or dark, motors are either turned on or off, and so forth. For such control situations the most significant aspect of the MCS®-51 architecture is its complete hardware support for one-bit, or *Boolean* variables (named in honor of Mathematician George Boole) as a separate data type.

The 8051 incorporates a number of special features which support the direct manipulation and testing of individual bits and allow the use of single-bit variables in performing logical operations. Taken together, these features are referred to as the MCS-51 *Boolean Processor*. While the bit-processing capabilities alone would be adequate to solve many control applications, their true power comes when they are used in conjunction with the microcomputer's byte-processing and numerical capabilities.

Many concepts embodied by the Boolean Processor will certainly be new even to experienced microcomputer system designers. The purpose of this Application Note is to explain these concepts and show how they are used.

For detailed information on these parts refer to the **Intel Microcontroller Handbook**, order number 210918. The instruction set, assembly language, and use of the 8051 assembler (ASM51) are further described in the **MCS®-51 Macro Assembler User's Guide for DOS Systems**, order number 122753.

Table 1. Features of Intel's Single-Chip Microcomputers

EPROM Program Memory	ROM Program Memory	External Program Memory	Program Memory (Int/Max)	Data Memory (Bytes)	Instr. Cycle Time	Input/Output Pins	Interrupt Sources	Reg. Banks
8748	8048	8035	1K 4K	64	2.5 μ s	27	2	2
—	8049	8039	2K 4K	128	1.36 μ s	27	2	2
8751	8051	8031	4K 64K	128	1.0 μ s	32	5	4

2.0 BOOLEAN PROCESSOR OPERATION

The Boolean Processing capabilities of the 8051 are based on concepts which have been around for some time. Digital computer systems of widely varying designs all have four functional elements in common (Figure 2):

- a central processor (CPU) with the control, timing, and logic circuits needed to execute stored instructions:
- a memory to store the sequence of instructions making up a program or algorithm:
- data memory to store variables used by the program:
- some means of communicating with the outside world.

The CPU usually includes one or more accumulators or special registers for computing or storing values during program execution. The instruction set of such a processor generally includes, at a minimum, operation classes to perform arithmetic or logical functions on program variables, move variables from one place to another, cause program execution to jump or conditionally branch based on register or variable states, and instructions to call and return from subroutines. The program and data memory functions sometimes share a single memory space, but this is not always the case. When the address spaces are separated, program and data memory need not even have the same basic word width.

A digital computer's flexibility comes in part from combining simple fast operations to produce more com-

plex (albeit slower) ones, which in turn link together eventually solving the problem at hand. A four-bit CPU executing multiple precision subroutines can, for example, perform 64-bit addition and subtraction. The subroutines could in turn be building blocks for floating-point multiplication and division routines. Eventually, the four-bit CPU can simulate a far more complex "virtual" machine.

In fact, *any* digital computer with the above four functional elements can (given time) complete *any* algorithm (though the proverbial room full of chimpanzees at word processors might first re-create Shakespeare's classics and this Application Note)! This fact offers little consolation to product designers who want programs to run as quickly as possible. By definition, a real-time control algorithm *must* proceed quickly enough to meet the preordained speed constraints of other equipment.

One of the factors determining how long it will take a microcomputer to complete a given chore is the number of instructions it must execute. What makes a given computer architecture particularly well- or poorly-suited for a class of problems is how well its instruction set matches the tasks to be performed. The better the "primitive" operations correspond to the steps taken by the control algorithm, the lower the number of instructions needed, and the quicker the program will run. All else being equal, a CPU supporting 64-bit arithmetic directly could clearly perform floating-point math faster than a machine bogged-down by multiple-precision subroutines. In the same way, direct support for bit manipulation naturally leads to more efficient programs handling the binary input and output conditions inherent in digital control problems.

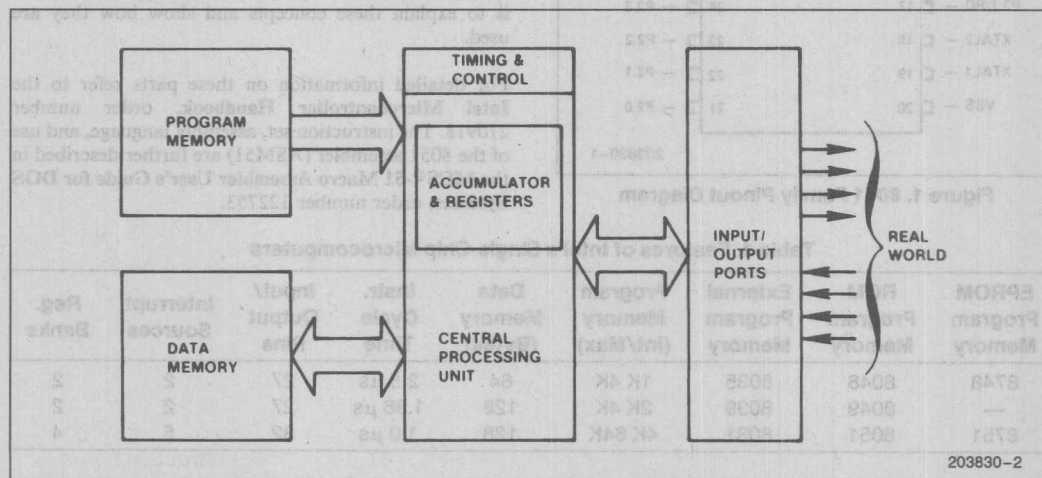


Figure 2. Block Diagram for Abstract Digital Computer

Processing Elements

The introduction stated that the 8051's bit-handling capabilities alone would be sufficient to solve some control applications. Let's see how the four basic elements of a digital computer—a CPU with associated registers, program memory, addressable data RAM, and I/O capability—relate to Boolean variables.

CPU. The 8051 CPU incorporates special logic devoted to executing several bit-wide operations. All told, there are 17 such instructions, all listed in Table 2. Not shown are 94 other (mostly byte-oriented) 8051 instructions.

Program Memory. Bit-processing instructions are fetched from the same program memory as other arithmetic and logical operations. In addition to the instruc-

tions of Table 2, several sophisticated program control features like multiple addressing modes, subroutine nesting, and a two-level interrupt structure are useful in structuring Boolean Processor-based programs.

Boolean instructions are one, two, or three bytes long, depending on what function they perform. Those involving only the carry flag have either a single-byte opcode or an opcode followed by a conditional-branch destination byte (Figure 3a). The more general instructions add a "direct address" byte after the opcode to specify the bit affected, yielding two or three byte encodings (Figure 3b). Though this format allows potentially 256 directly addressable bit locations, not all of them are implemented in the 8051 family.

Table 2. MCS-51™ Boolean Processing Instruction Subset

Mnemonic	Description	Byte	Cyc
SETB C	Set Carry flag	1	1
SETB bit	Set direct Bit	2	1
CLR C	Clear Carry flag	1	1
CLR bit	Clear direct bit	2	1
CPL C	Complement Carry flag	1	1
CPL bit	Complement direct bit	2	1
MOV C.bit	Move direct bit to Carry flag	2	1
MOV bit.C	Move Carry flag to direct bit	2	2
ANL C.bit	AND direct bit to Carry flag	2	2
ANL C.bit	AND complement of direct bit to Carry flag	2	2
ORL C.bit	OR direct bit to Carry flag	2	2
ORL C.bit	OR complement of direct bit to Carry flag	2	2
JC rel	Jump if Carry is flag is set	2	2
JNC rel	Jump if No Carry flag	2	2
JB bit.rel	Jump if direct Bit set	3	2
JNB bit.rel	Jump if direct Bit Not set	3	2
JBC bit.rel	Jump if direct Bit is set & Clear bit	3	2

Address mode abbreviations

C—Carry flag.

bit—128 software flags, any I/O pin, control or status bit.

rel—All conditional jumps include an 8-bit offset byte. Range is +127 – 128 bytes relative to first byte of the following instruction.

All mnemonics copyrighted © Intel Corporation 1980.

opcode

SETB C
CLR C
CPL C

opcode

displacement

JC rel
JNC rel

a.) Carry Control and Test Instructions

opcode

bit address

SETB bit
CLR bit
CPL bit
ANL C, bit
ANL C,/ bit
ORL C, bit
ORL C,/ bit
MOV C, bit
MOV bit,C

opcode

bit address

displacement

JB bit, rel
JNB bit, rel
JBC bit, rel

b.) Bit Manipulation and Test Instructions

Figure 3. Bit Addressing Instruction Formats

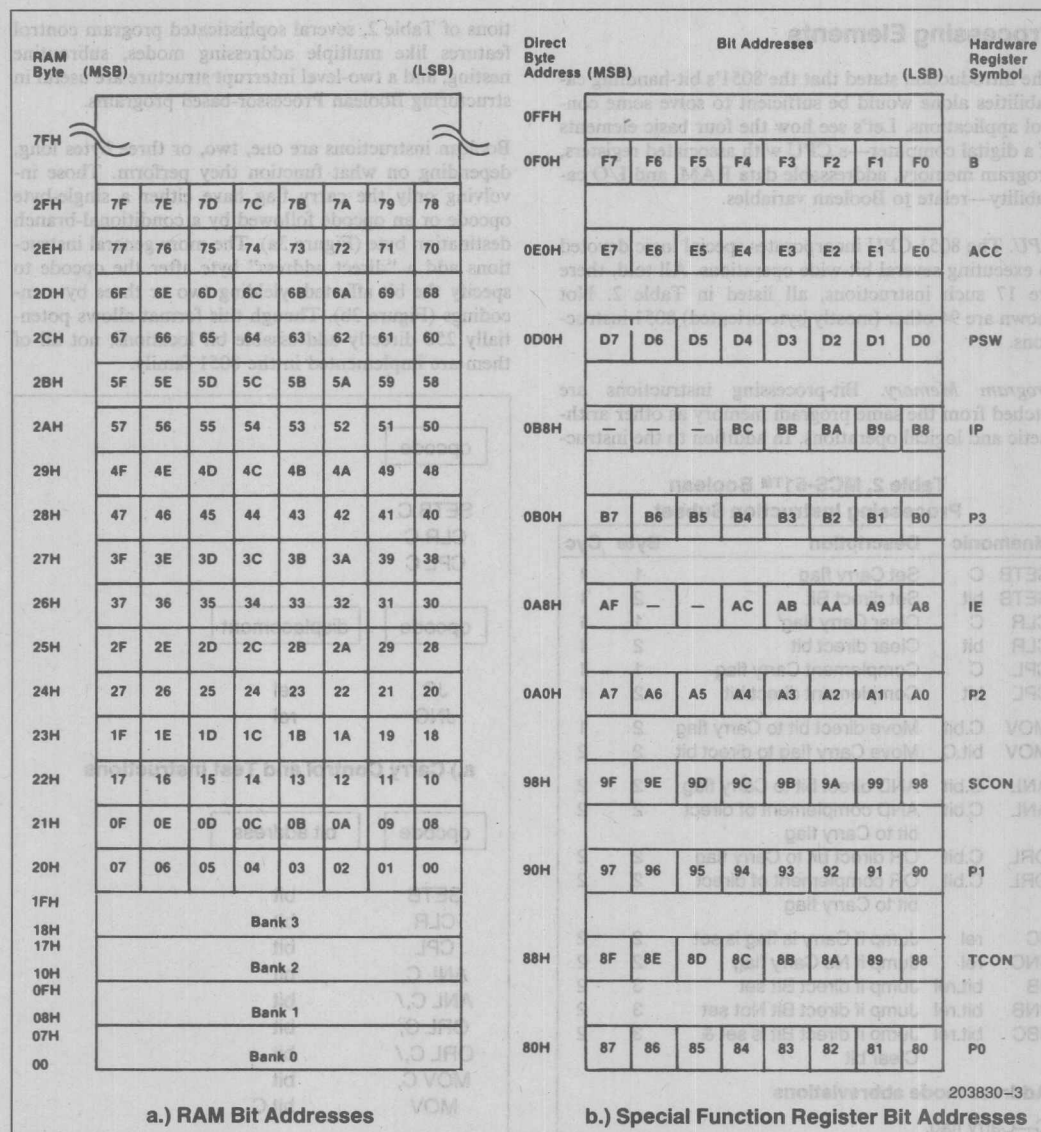


Figure 4. Bit Address Maps

Data Memory. The instructions in Figure 3b can operate directly upon 144 general purpose bits forming the Boolean processor "RAM." These bits can be used as software flags or to store program variables. Two operand instructions use the CPU's carry flag ("C") as a special one-bit register: in a sense, the carry is a "Boolean accumulator" for logical operations and data transfers.

Input/Output. All 32 I/O pins can be addressed as individual inputs, outputs, or both, in any combination. Any pin can be a control strobe output, status (Test) input, or serial I/O link implemented via software. An additional 33 individually addressable bits reconfigure, control, and monitor the status of the CPU and all on-chip peripheral functions (timer counters, serial port modes, interrupt logic, and so forth).

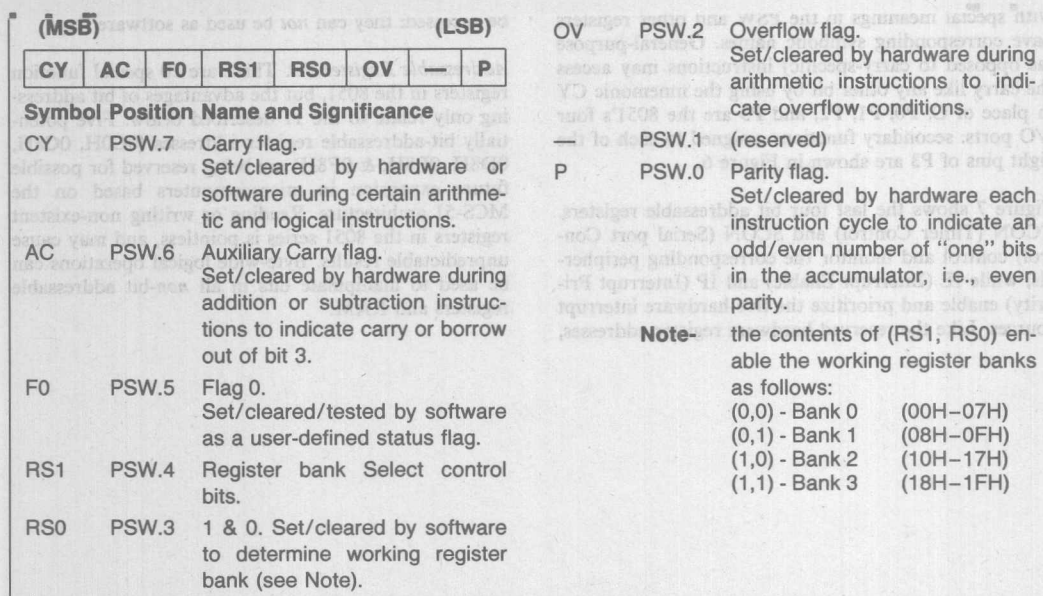


Figure 5. PSW—Program Status Word Organization

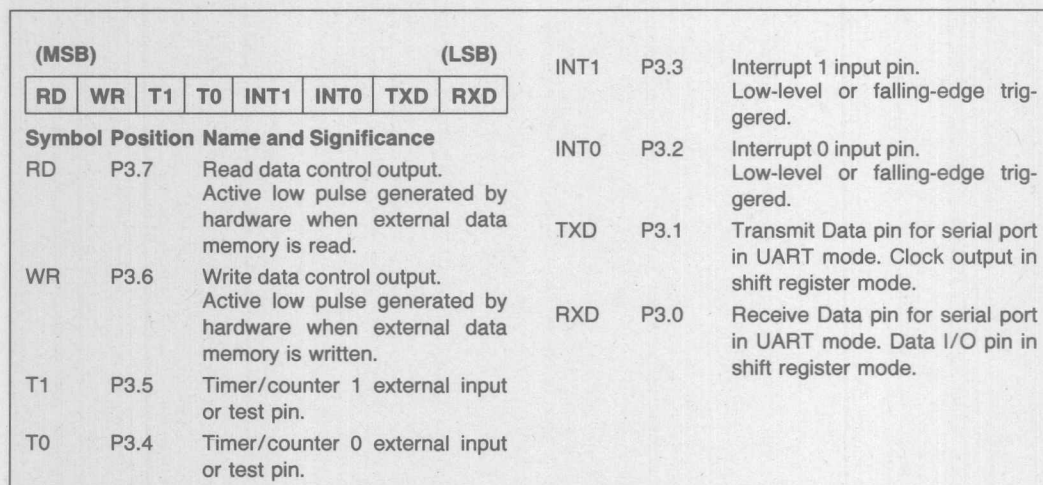


Figure 6. P3—Alternate I/O Functions of Port 3

Direct Bit Addressing

The most significant bit of the direct address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 32 bytes of on-chip RAM, between RAM addresses 20H and 2FH (Figure 4a). They are numbered consecutively from the lowest-order byte's lowest-order bit through the highest-order byte's highest-order bit.

Bit addresses between 128 and 255 (80H and 0FFH) correspond to bits in a number of special registers, mostly used for I/O or peripheral control. These positions are numbered with a different scheme than RAM: the five high-order address bits match those of the register's own address, while the three low-order bits identify the bit position within that register (Figure 4b).

Notice the column labeled "Symbol" in Figure 5. Bits with special meanings in the PSW and other registers have corresponding symbolic names. General-purpose (as opposed to carry-specific) instructions may access the carry like any other bit by using the mnemonic CY in place of C, P0, P1, P2, and P3 are the 8051's four I/O ports: secondary functions assigned to each of the eight pins of P3 are shown in Figure 6.

Figure 7 shows the last four bit addressable registers. TCON (Timer Control) and SCON (Serial port Control) control and monitor the corresponding peripherals, while IE (Interrupt Enable) and IP (Interrupt Priority) enable and prioritize the five hardware interrupt sources. Like the reserved hardware register addresses,

the five bits not implemented in IE and IP should not be accessed: they can *not* be used as software flags.

Addressable Register Set. There are 20 special function registers in the 8051, but the advantages of bit addressing only relate to the 11 described below. Five potentially bit-addressable register addresses (0C0H, 0C8H, 0D8H, 0E8H, & 0F8H) are being reserved for possible future expansion in microcomputers based on the MCS-51 architecture. Reading or writing non-existent registers in the 8051 series is pointless, and may cause unpredictable results. Byte-wide logical operations can be used to manipulate bits in all *non*-bit addressable registers and RAM.

PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
1 & 0. Self-cleared by software to determine working register bank (see 140a).	Register bank Select control bit.	as a user-defined status flag. Self-cleared/forced by software out of bit 3.	Flag 0.	as follows:	able the working register banks
				(1.0) - Bank 3 (18H-1FH)	
				(1.0) - Bank 2 (10H-17H)	
				(0.1) - Bank 1 (08H-0FH)	
				(0.0) - Bank 0 (00H-07H)	

Figure 5. PSW—Program Status Word Organization

(MSB)	(LSB)	Symbol Position Name and Significance	RD	WR	TI	T0	INT1	INT0	TxD	RxD
P3.7	P3.0	Read data control output. Active low pulses generated by hardware when external data memory is read.								
P3.6	P3.0	Write data control output. Active low pulses generated by hardware when external data memory is written.								
P3.5	P3.0	Timer/counter 1 external input or fast pin.								
P3.4	P3.0	Timer/counter 0 external input or fast pin.								
P3.3	P3.0	Transmit Data pin for serial port in UART mode. Clock output in shift register mode.								
P3.2	P3.0	Receive Data pin for serial port in UART mode. Data I/O pin in shift register mode.								
P3.1	P3.0	Interrupt 1 input pin. Low-level or falling-edge triggered.								
P3.0	P3.0	Interrupt 0 input pin. Low-level or falling-edge triggered.								

Figure 6. P3—Alternate I/O Functions of Port 3

Bit addresses between 128 and 255 (80H and 0FFH) correspond to bits in a number of special registers mostly used for I/O or peripheral control. These positions are numbered with a different scheme than RAM: the five high-order address bits match those of the register's own address, while the three low-order bits identify the bit position within that register (Figure 4b).

Direct Bit Addressing

The most significant bit of the direct address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 32 bytes of on-chip RAM between RAM addresses 20H and 2FH (Figure 4a). They are numbered consecutively from the lowest-order byte's lowest-order bit through the highest-order byte's highest-order bit.

(MSB)				(LSB)			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol Position Name and Significance

TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.

a.) TCON—Timer/Counter Control/Status Register

(MSB)				(LSB)			
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Symbol Position Name and Significance

SM0	SCON.7	Serial port Mode control bit 0. Set/cleared by software (see note).
SM1	SCON.6	Serial port Mode control bit 1. Set/cleared by software (see note).
SM2	SCON.5	Serial port Mode control bit 2. Set by software to disable reception of frames for which bit 8 is zero.
REN	SCON.4	Receiver Enable control bit. Set/cleared by software to enable/disable serial data reception.
TB8	SCON.3	Transmit Bit 8. Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.

b.) SCON—Serial Port Control/Status Register

IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.
IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT0	TCON.0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

RB8	SCON.2	Receive Bit 8. Set/cleared by hardware to indicate state of ninth data bit received.
TI	SCON.1	Transmit Interrupt flag. Set by hardware when byte transmitted. Cleared by software after servicing.
RI	SCON.0	Receive Interrupt flag. Set by hardware when byte received. Cleared by software after servicing.

Note— the state of (SM0, SM1) selects:

(0,0)	—Shift register I/O expansion.
(0,1)	—8-bit UART, variable data rate.
(1,0)	—9-bit UART, fixed data rate.
(1,1)	—9-bit UART, variable data rate.

Figure 7. Peripheral Configuration Registers

(MSB)				(LSB)			
EA	—	—	ES	ET1	EX1	ET1	EX0
Symbol Position Name and Significance							
EA	IE.7	Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4–IE.0.					
—	IE.6	(reserved)					
—	IE.5	(reserved)					
ES	IE.4	Enable Serial port control bit. Set/cleared by software to enable/disable interrupts from TI or RI flags.					
ET1	IE.3	Enable Timer 1 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 1.					

c.) IE—Interrupt Enable Register

(MSB)				(LSB)			
—	—	—	PS	PT1	PX1	PT0	PX0
Symbol Position Name and Significance							
—	IP.7	(reserved)					
—	IP.6	(reserved)					
—	IP.5	(reserved)					
PS	IP.4	Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.					
PT1	IP.3	Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.					

d.) IP—Interrupt Priority Control Register

EX1	IE.2	Enable External interrupt 1 control bit. Set/cleared by software to enable/disable interrupts from INT1.
ET0	IE.1	Enable Timer 0 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 0.
EX0	IE.0	Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INTO.

PX1	IP.2	External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.
PT0	IP.1	Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0.
PX0	IP.0	External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INTO.

Figure 7. Peripheral Configuration Registers (Continued)

The accumulator and B registers (A and B) are normally involved in byte-wide arithmetic, but their individual bits can also be used as 16 general software flags. Added with the 128 flags in RAM, this gives 144 general purpose variables for bit-intensive programs. The program status word (PSW) in Figure 5 is a collection of flags and machine status bits including the carry flag itself. Byte operations acting on the PSW can therefore affect the carry.

Instruction Set

Having looked at the bit variables available to the Boolean Processor, we will now look at the four classes of

instructions that manipulate these bits. It may be helpful to refer back to Table 2 while reading this section.

State Control. Addressable bits or flags may be set, cleared, or logically complemented in one instruction cycle with the two-byte instructions SETB, CLR, and CPL. (The "B" affixed to SETB distinguishes it from the assembler "SET" directive used for symbol definition.) SETB and CLR are analogous to loading a bit with a constant: 1 or 0. Single byte versions perform the same three operations on the carry.

The MCS-51 assembly language specifies a bit address in any of three ways:

- by a number or expression corresponding to the direct bit address (0–255):

- by the name or address of the register containing the bit, the *dot operator* symbol (a period: "."), and the bit's position in the register (7-0):
- in the case of control and status registers, by the predefined assembler symbols listed in the first columns of Figures 5-7.

Bits may also be given user-defined names with the assembler "BIT" directive and any of the above techniques. For example, bit 5 of the PSW may be cleared by any of the four instructions.

```

USR_FLG BIT PSW.5 ; User Symbol Definition
...
CLR OD5H ; Absolute Addressing
CLR PSW.5 ; Use of Dot Operator
CLR FO ; Pre-Defined Assembler
; Symbol
CLR USR_FLG ; User-Defined Symbol

```

Data Transfers. The two-byte MOV instructions can transport any addressable bit to the carry in one cycle, or copy the carry to the bit in two cycles. A bit can be moved between two arbitrary locations via the carry by combining the two instructions. (If necessary, push and pop the PSW to preserve the previous contents of the carry.) These instructions can replace the multi-instruction sequence of Figure 8, a program structure appearing in controller applications whenever flags or outputs are conditionally switched on or off.

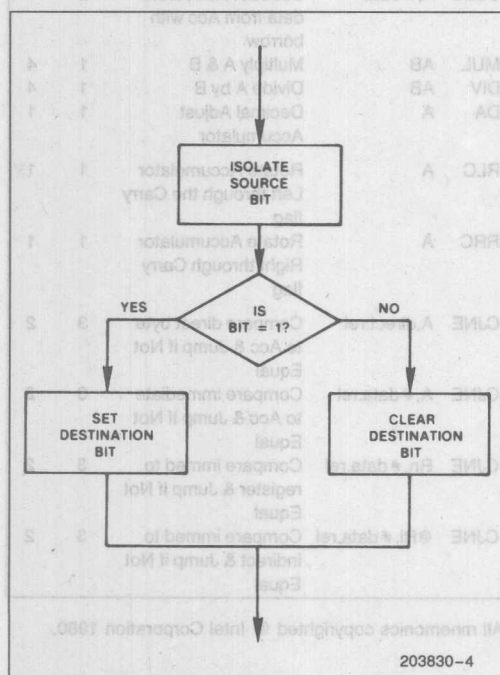


Figure 8. Bit Transfer Instruction Operation

Logical Operations. Four instructions perform the logical-AND and logical-OR operations between the carry and another bit, and leave the results in the carry. The instruction mnemonics are ANL and ORL; the absence or presence of a slash mark ("/") before the source operand indicates whether to use the positive-logic value or the logical complement of the addressed bit. (The source operand itself is never affected.)

Bit-test Instructions. The conditional jump instructions "JC rel" (Jump on Carry) and "JNC rel" (Jump on Not Carry) test the state of the carry flag, branching if it is a one or zero, respectively. (The letters "rel" denote relative code addressing.) The three-byte instructions "JB bit.rel" and "JNB bit.rel" (Jump on Bit and Jump on Not Bit) test the state of any addressable bit in a similar manner. A fifth instruction combines the Jump on Bit and Clear operations. "JBC bit.rel" conditionally branches to the indicated address, then clears the bit in the same two cycle instruction. This operation is the same as the MCS-48 "JTF" instructions.

All 8051 conditional jump instructions use program counter-relative addressing, and all execute in two cycles. The last instruction byte encodes a signed displacement ranging from -128 to +127. During execution, the CPU adds this value to the incremented program counter to produce the jump destination. Put another way, a conditional jump to the immediately following instruction would encode 00H in the offset byte.

A section of program or subroutine written using only relative jumps to nearby addresses will have the same machine code independent of the code's location. An assembled routine may be repositioned anywhere in memory, even crossing memory page boundaries, without having to modify the program or recompute destination addresses. To facilitate this flexibility, there is an unconditional "Short Jump" (SJMP) which uses relative addressing as well. Since a programmer would have quite a chore trying to compute relative offset values from one instruction to another, ASM51 automatically computes the displacement needed given only the destination address or label. An error message will alert the programmer if the destination is "out of range."

The so-called "Bit Test" instructions implemented on many other microprocessors simply perform the logical-AND operation between a byte variable and a constant mask, and set or clear a zero flag depending on the result. This is essentially equivalent to the 8051 "MOV C.bit" instruction. A second instruction is then needed to conditionally branch based on the state of the zero flag. This does *not* constitute abstract bit-addressing in the MCS-51 sense. A flag exists only as a field

within a register: to reference a bit the programmer must know and specify both the encompassing register and the bit's position therein. This constraint severely limits the flexibility of symbolic bit addressing and reduces the machine's code-efficiency and speed.

Interaction with Other Instructions. The carry flag is also affected by the instructions listed in Table 3. It can be rotated through the accumulator, and altered as a side effect of arithmetic instructions. Refer to the User's Manual for details on how these instructions operate.

Simple Instruction Combinations

By combining general purpose bit operations with certain addressable bits, one can "custom build" several hundred useful instructions. All eight bits of the PSW can be tested directly with conditional jump instructions to monitor (among other things) parity and overflow status. Programmers can take advantage of 128 software flags to keep track of operating modes, resource usage, and so forth.

The Boolean instructions are also the most efficient way to control or reconfigure peripheral and I/O registers. All 32 I/O lines become "test pins," for example, tested by conditional jump instructions. Any output pin can be toggled (complemented) in a single instruction cycle. Setting or clearing the Timer Run flags (TR0 and TR1) turn the timer/counters on or off; polling the same flags elsewhere lets the program determine if a timer is running. The respective overflow flags (TF0 and TF1) can be tested to determine when the desired period or count has elapsed, then cleared in preparation for the next repetition. (For the record, these bits are all part of the TCON register, Figure 7a. Thanks to symbolic bit addressing, the programmer only needs to remember the mnemonic associated with each function. In other words, don't bother memorizing control word layouts.)

In the MCS-48 family, instructions corresponding to some of the above functions require specific opcodes. Ten different opcodes serve to clear/complement the software flags F0 and F1, enable/disable each interrupt, and start/stop the timer. In the 8051 instruction set, just three opcodes (SETB, CLR, CPL) with a direct bit address appended perform the same functions. Two test instructions (JB and JNB) can be combined with bit addresses to test the software flags, the 8048 I/O pins T0, T1, and INT, and the eight accumulator bits, replacing 15 more different instructions. Table 4a shows how 8051 programs implement software flag and machine control functions associated with special opcodes in the 8048. In every case the MCS-51 solution requires the same number of machine cycles, and executes 2.5 times faster.

Table 3. Other Instructions Affecting the Carry Flag

Mnemonic	Description	Byte	Cyc
ADD A,Rn	Add register to Accumulator	1	1
ADD A,direct	Add direct byte to Accumulator	2	1
ADD A,@Ri	Add indirect RAM to Accumulator	1	1
ADD A,#data	Add immediate data to Accumulator	2	1
ADDC A,Rn	Add register to Accumulator with Carry flag	1	1
ADDC A,direct	Add direct byte to Accumulator with Carry flag	2	1
ADDC A,@Ri	Add indirect RAM to Accumulator with Carry flag	1	1
ADDC A,#data	Add immediate data to Acc with Carry flag	2	1
SUBB A,Rn	Subtract register from Accumulator with borrow	1	1
SUBB A,direct	Subtract direct byte from Acc with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from Acc with borrow	1	1
SUBB A,#data	Subtract immediate data from Acc with borrow	2	1
MUL AB	Multiply A & B	1	4
DIV AB	Divide A by B	1	4
DA A	Decimal Adjust Accumulator	1	1
RLC A	Rotate Accumulator Left through the Carry flag	1	1
RRC A	Rotate Accumulator Right through Carry flag	1	1
CJNE A,direct.rel	Compare direct byte to Acc & Jump if Not Equal	3	2
CJNE A,#data.rel	Compare immediate to Acc & Jump if Not Equal	3	2
CJNE Rn,#data.rel	Compare immed to register & Jump if Not Equal	3	2
CJNE @Ri,#data.rel	Compare immed to indirect & Jump if Not Equal	3	2

All mnemonics copyrighted © Intel Corporation 1980.

Table 4a. Contrasting 8048 and 8051 Bit Control and Testing Instructions

8048 Instruction	Bytes	Cycles	μSec	8x51 Instruction	Bytes	Cycles & μSec
Flag Control						
CLR C	1	1	2.5	CLR C	1	1
CPL F0	1	1	2.5	CPL F0	2	1
Flag Testing						
JNC offset	2	2	5.0	JNC rel	2	2
JF0 offset	2	2	5.0	JB F0.rel	3	2
JB7 offset	2	2	5.0	JB ACC.7.rel	3	2
Peripheral Polling						
JT0 offset	2	2	5.0	JB T0.rel	3	2
JN1 offset	2	2	5.0	JNB INT0.rel	3	2
JTF offset	2	2	5.0	JBC TF0.rel	3	2
Machine and Peripheral Control						
STRT T	1	1	2.5	SETB TR0	2	1
EN 1	1	1	2.5	SETB EX0	2	1
DIS TCNT1	1	1	2.5	CLR ET0	2	1

2

Table 4b. Replacing 8048 Instruction Sequences with Single 8x51 Instructions

8048 Instruction	Bytes	Cycles	μSec	8051 Instruction	Bytes	Cycles & μSec
Flag Control						
Set carry						
CLR C				SETB C	1	1
CPL C	= 2	2	5.0			
Set Software Flag				SETB F0	2	1
CLR F0						
CPL F0	= 2	2	5.0			
Turn Off Output Pin						
ANL P1.#0FBH	= 2	2	5.0	CLR P1.2	2	1
Complement Output Pin						
IN A.P1						
XRL A.#04H				CPL P1.2	2	1
OUTL P1.A	= 4	6	15.0			
Clear Flag in RAM						
MOV R0.#FLGADR				CLR USER_FLG	2	1
MOV A.@R0						
ANL A.#FLGMASK						
MOV @R0.A	= 6	6	15.0			

Table 4b. Replacing 8048 Instruction Sequences with Single 8x51 Instructions (Continued)

8048 Instruction	Bytes	Cycles	μSec	8x51 Instruction	Bytes	Cycles & μSec
Flag Testing:						
Jump if Software Flag is 0						
JF0 \$+4						
JMP offset = 4		4	10.0	JNB F0.rel	3	2
Jump if Accumulator bit is 0						
CPL A						
JB7 offset						
CPL A = 4		4	10.0	JNB ACC.7.rel	3	2
Peripheral Polling						
Test if Input Pin is Grounded						
IN A.P1						
CPL A						
JB3 offset = 4		5	12.5	JNB P1.3.rel	3	2
Test if Interrupt Pin is High						
JN1 \$+4						
JMP offset = 4		4	10.0	JB INT0.rel	3	2

3.0 BOOLEAN PROCESSOR APPLICATIONS

So what? Then what does all this buy you?

Qualitatively, nothing. All the same capabilities could be (and often have been) implemented on other machines using awkward sequences of other basic operations. As mentioned earlier, any CPU can solve any problem given enough time.

Quantitatively, the differences between a solution allowed by the 8051 and those required by previous architectures are numerous. What the 8051 Family buys you is a faster, cleaner, lower-cost solution to micro-controller applications.

The opcode space freed by condensing many specific 8048 instructions into a few general operations has been used to add new functionality to the MCS-51 architecture—both for byte and bit operations. 144 software flags replace the 8048's two. These flags (and the carry) may be directly set, not just cleared and complemented, and all can be tested for either state, not just one. Operating mode bits previously inaccessible may be read, tested, or saved. Situations where the 8051 instruction set provides new capabilities are contrasted with 8048 instruction sequences in Table 4b. Here the 8051 speed advantage ranges from 5x to 15x!

Combining Boolean and byte-wide instructions can produce great synergy. An MCS-51 based application will prove to be:

- simpler to write since the architecture correlates more closely with the problems being solved;
- easier to debug because more individual instructions have no unexpected or undesirable side-effects;
- more byte efficient due to direct bit addressing and program counter relative branching;
- faster running because fewer bytes of instruction need to be fetched and fewer conditional jumps are processed;
- lower cost because of the high level of system-integration within one component.

These rather unabashed claims of excellence shall not go unsubstantiated. The rest of this chapter examines less trivial tasks simplified by the Boolean processor. The first three compare the 8051 with other micro-processors; the last two go into 8051-based system designs in much greater depth.

Design Example # 1—Bit Permutation

First off, we'll use the bit-transfer instructions to permute a lengthy pattern of bits.

A steadily increasing number of data communication products use encoding methods to protect the security of sensitive information. By law, interstate financial transactions involving the Federal banking system must be transmitted using the Federal Information Processing *Data Encryption Standard* (DES).

Basically, the DES combines eight bytes of "plaintext" data (in binary, ASCII, or any other format) with a 56-bit "key", producing a 64-bit encrypted value for transmission. At the receiving end the same algorithm is applied to the incoming data using the same key, reproducing the original eight byte message. The algorithm used for these permutations is fixed; different user-defined keys ensure data privacy.

It is not the purpose of this note to describe the DES in any detail. Suffice it to say that encryption/decryption is a long, iterative process consisting of rotations, exclusive -OR operations, function table look-ups, and an extensive (and quite bizarre) sequence of bit permutation, packing, and unpacking steps. (For further details refer to the June 21, 1979 issue of *Electronics* magazine.) The bit manipulation steps are included, it is rumored, to impede a general purpose digital supercomputer trying to "break" the code. Any algorithm implementing the DES with previous generation microprocessors would spend virtually all of its time diddling bits.

The bit manipulation performed is typified by the Key Schedule Calculation represented in Figure 9. This step is repeated 16 times for each key used in the course of a transmission. In essence, a seven-byte, 56-bit "Shifted Key Buffer" is transformed into an eight-byte, "Permutation Buffer" without altering the shifted Key. The arrows in Figure 9 indicate a few of the translation steps. Only six bits of each byte of the Permutation Buffer are used; the two high-order bits of each byte are cleared. This means only 48 of the 56 Shifted Key Buffer bits are used in any one iteration.

Different microprocessor architectures would best implement this type of permutation in different ways. Most approaches would share the steps of Figure 10a:

- Initialize the Permutation Buffer to default state (ones or zeroes):
- Isolate the state of a bit of a byte from the Key Buffer. Depending on the CPU, this might be accomplished by rotating a word of the Key Buffer through a carry flag or testing a bit in memory or an accumulator against a mask byte:
- Perform a conditional jump based on the carry or zero flag if the Permutation Buffer default state is correct:
- Otherwise reverse the corresponding bit in the permutation buffer with logical operations and mask bytes.

Each step above may require several instructions. The last three steps must be repeated for all 48 bits. Most microprocessors would spend 300 to 3,000 microseconds on each of the 16 iterations.

Notice, though, that this flow chart looks a lot like Figure 8. The Boolean Processor can permute bits by simply moving them from the source to the carry to the destination—a total of two instructions taking four bytes and three microseconds per bit. Assume the Shifted Key Buffer and Permutation Buffer both reside in bit-addressable RAM, with the bits of the former assigned symbolic names SKB_1, SKB_2, . . . SKB_56, and that the bytes of the latter are named PB_1, . . . PB_8. Then working from Figure 9, the software for the permutation algorithm would be that of Example 1a. The total routine length would be 192 bytes, requiring 144 microseconds.

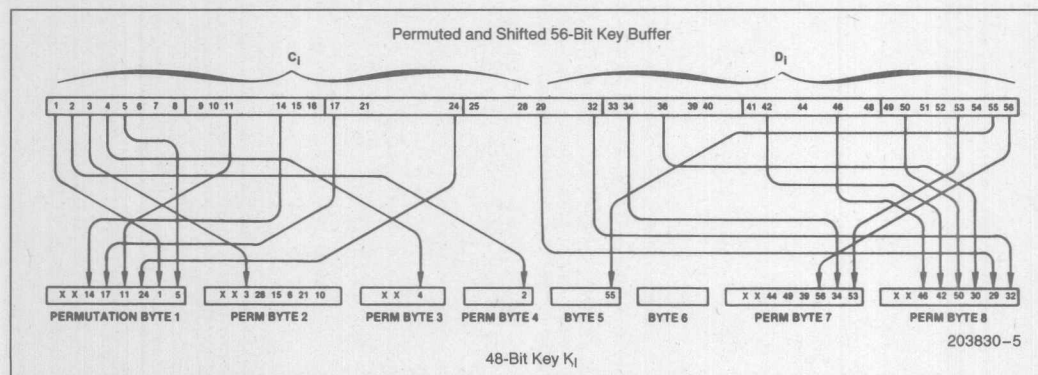


Figure 9. DES Key Schedule Transformation

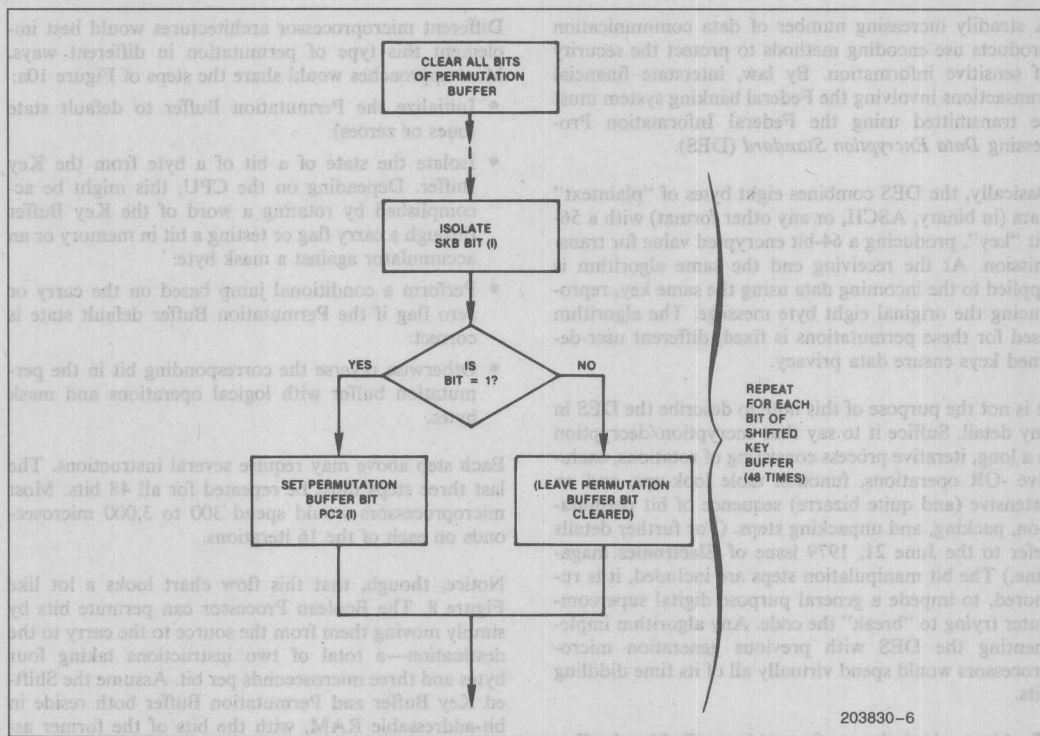


Figure 10a. Flowchart for Key Permutation Attempted with a Byte Processor

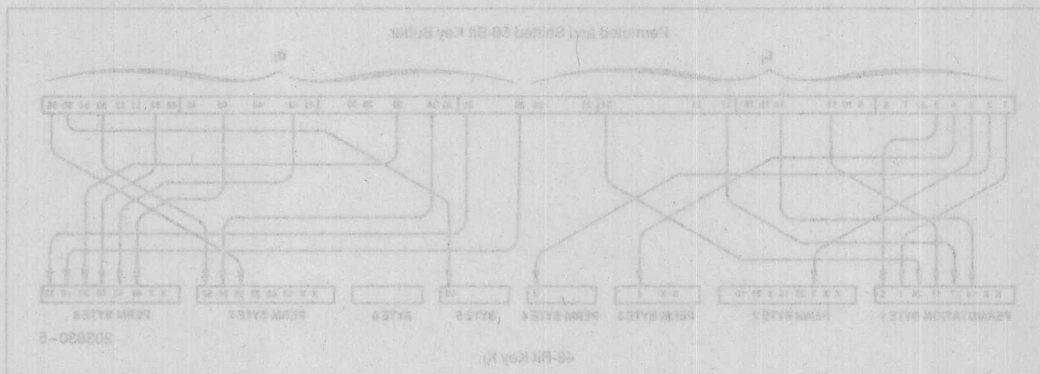


Figure 9. DES Key Schedule Transformation

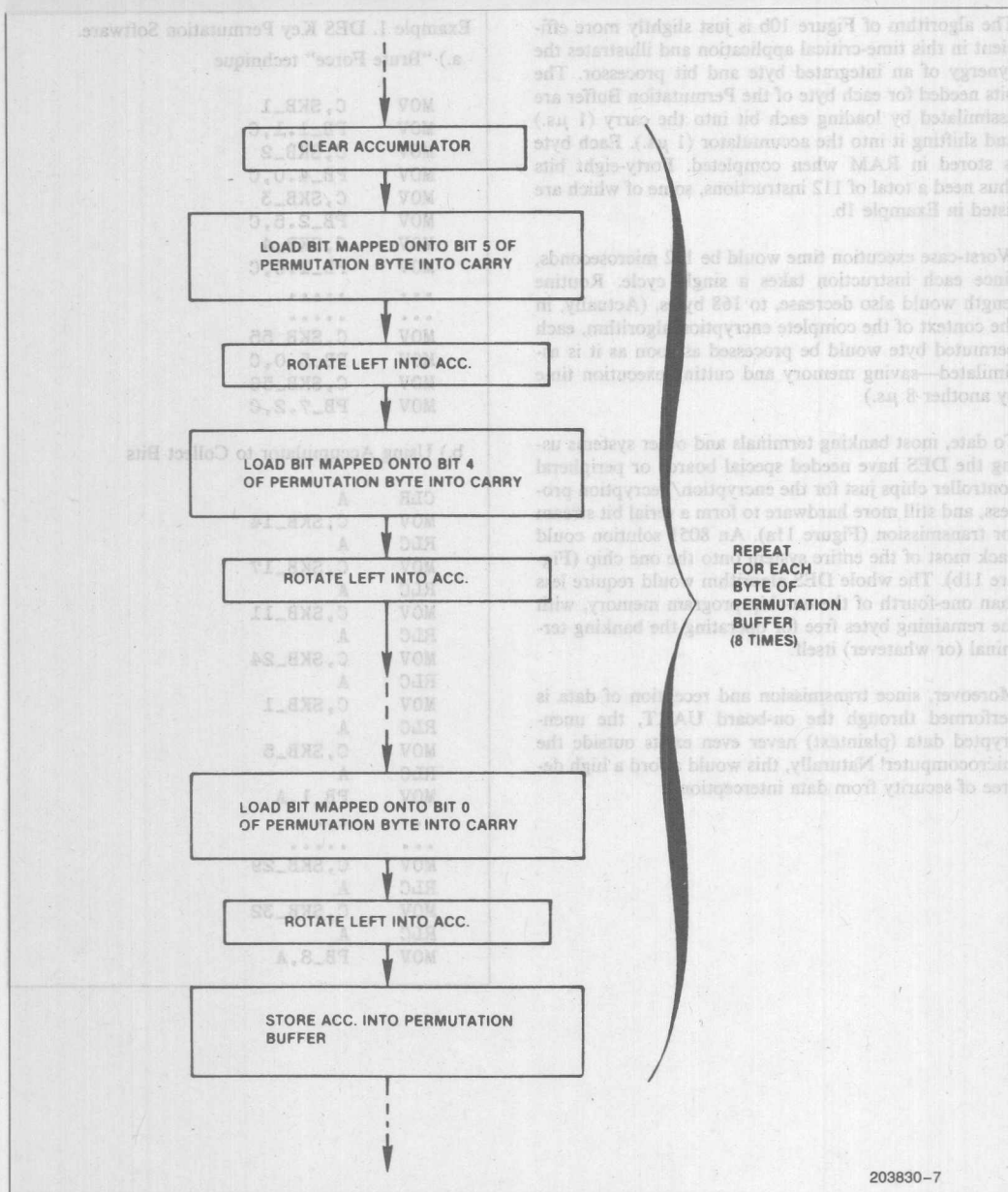


Figure 10b. DES Key Permutation with Boolean Processor

The algorithm of Figure 10b is just slightly more efficient in this time-critical application and illustrates the synergy of an integrated byte and bit processor. The bits needed for each byte of the Permutation Buffer are assimilated by loading each bit into the carry (1 μ s.) and shifting it into the accumulator (1 μ s.). Each byte is stored in RAM when completed. Forty-eight bits thus need a total of 112 instructions, some of which are listed in Example 1b.

Worst-case execution time would be 112 microseconds, since each instruction takes a single cycle. Routine length would also decrease, to 168 bytes. (Actually, in the context of the complete encryption algorithm, each permuted byte would be processed as soon as it is assimilated—saving memory and cutting execution time by another 8 μ s.)

To date, most banking terminals and other systems using the DES have needed special boards or peripheral controller chips just for the encryption/decryption process, and still more hardware to form a serial bit stream for transmission (Figure 11a). An 8051 solution could pack most of the entire system onto the one chip (Figure 11b). The whole DES algorithm would require less than one-fourth of the on-chip program memory, with the remaining bytes free for operating the banking terminal (or whatever) itself.

Moreover, since transmission and reception of data is performed through the on-board UART, the unencrypted data (plaintext) never even exists outside the microcomputer! Naturally, this would afford a high degree of security from data interception.

Example 1. DES Key Permutation Software.

a.) "Brute Force" technique

```
MOV    C,SKB_1
MOV    PB_1.1,C
MOV    C,SKB_2
MOV    PB_4.0,C
MOV    C,SKB_3
MOV    PB_2.5,C
MOV    C,SKB_4
MOV    PB_1.0,C
...
...
MOV    C,SKB_55
MOV    PB_5.0,C
MOV    C,SKB_56
MOV    PB_7.2,C
```

b.) Using Accumulator to Collect Bits

```
CLR    A
MOV    C,SKB_14
RLC    A
MOV    C,SKB_17
RLC    A
MOV    C,SKB_11
RLC    A
MOV    C,SKB_24
RLC    A
MOV    C,SKB_1
RLC    A
MOV    C,SKB_5
RLC    A
MOV    PB_1,A
...
...
MOV    C,SKB_29
RLC    A
MOV    C,SKB_32
RLC    A
MOV    PB_8,A
```

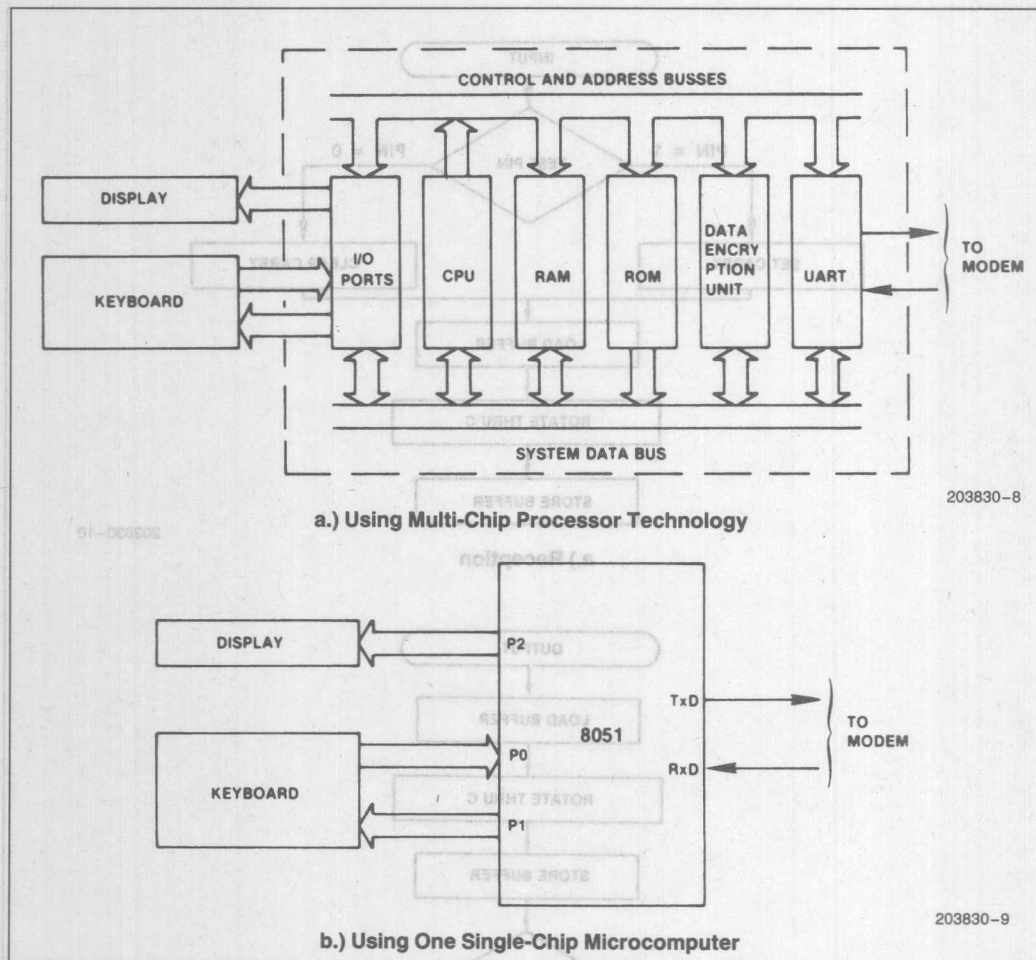


Figure 11. Secure Banking Terminal Block Diagram

Design Example #2—Software Serial I/O

An exercise often imposed on beginning microcomputer students is to write a program simulating a UART. Though doing this with the 8051 Family may appear to be a moot point (given that the hardware for a full UART is on-chip), it is still instructive to see how it would be done, and maintains a product line tradition.

As it turns out, the 8051 microcomputers can receive or transmit serial data via software very efficiently using the Boolean instruction set. Since any I/O pin may be a serial input or output, several serial links could be maintained at once.

Figures 12a and 12b show algorithms for receiving or transmitting a byte of data. (Another section of program would invoke this algorithm eight times, synchronizing it with a start bit, clock signal, software delay, or timer interrupt.) Data is received by testing an input pin, setting the carry to the same state, shifting the carry into a data buffer, and saving the partial frame in internal RAM. Data is transmitted by shifting an output buffer through the carry, and generating each bit on an output pin.

A side-by-side comparison of the software for this common "bit-banging" application with three different microprocessor architectures is shown in Table 5a and 5b. The 8051 solution is more efficient than the others on every count!

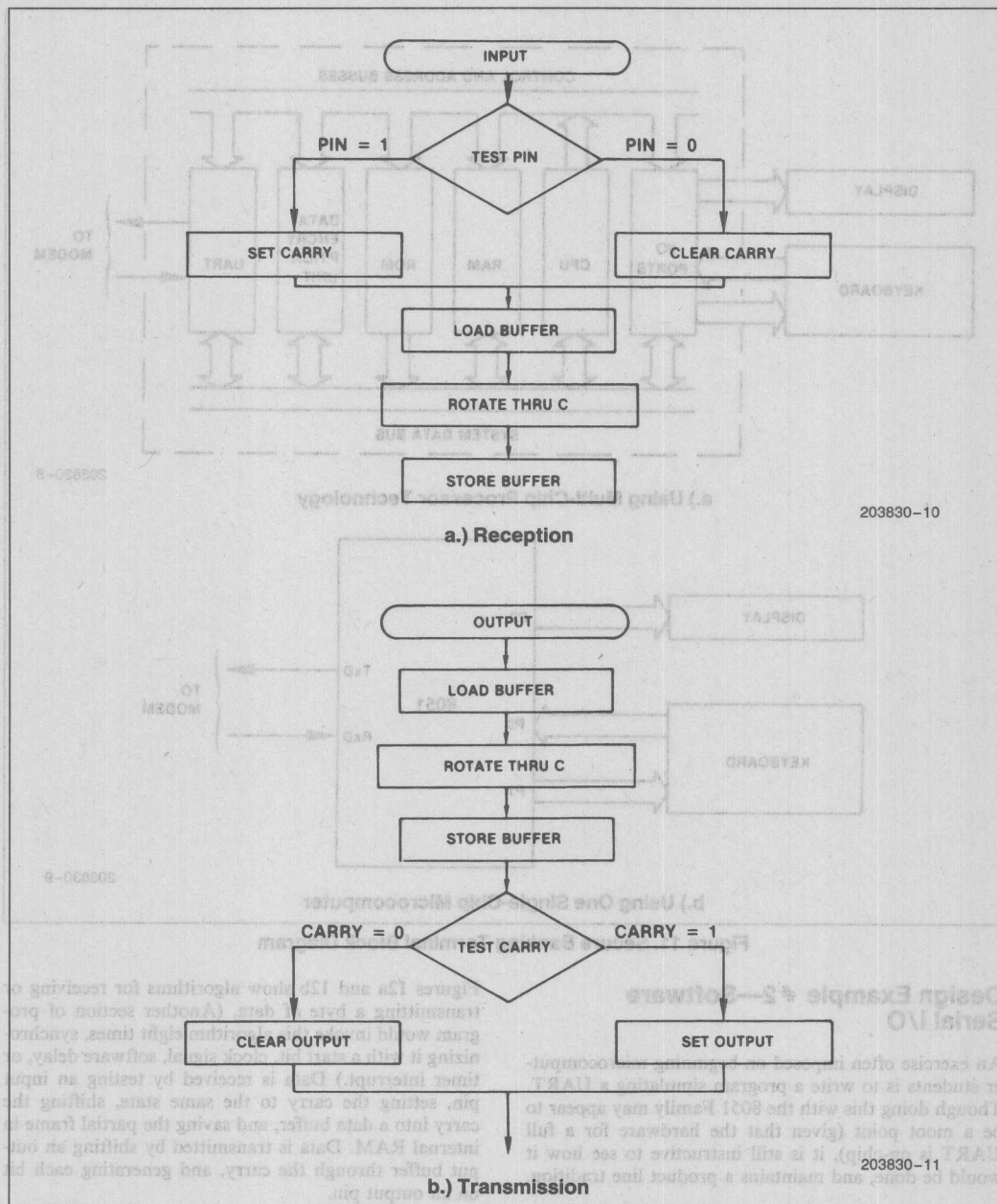


Figure 12. Serial I/O Algorithms

Table 5. Serial I/O Programs for Various Microprocessors

a.) Input Routine.		
8085	8048	8051
IN SERPORT	CLR C	MOV C,SERPIN
ANI MASK	JNT0 I.O	
JZ I.O	CPL C	
CMC	MOV R0,#SERBUF	
I.O: LXI HL,SERBUF	MOV A,@R0	MOV A,SERBUF
MOV A,M	RRC A	RRC A
RR	MOV @R0,A	MOV SERBUF,A
MOV M,A		
RESULTS:		
8 INSTRUCTIONS	7 INSTRUCTIONS	4 INSTRUCTIONS
14 BYTES	9 BYTES	7 BYTES
56 STATES	9 CYCLES	4 CYCLES
19 uSEC.	22.5 uSEC.	4 uSEC.
b.) Output Routine.		
8085	8048	8051
LXI HL,SERBUF	MOV R0,#SERBUF	
MOV A,M	MOV A,@R0	MOV A,SERBUF
RR	RRC A	RRC A
MOV M,A	MOV @R0,A	MOV SERBUF,A
IN SERPORT		
JC HI	JC HI	
I.O: ANI NOT MASK	ANI SERPRT,#NOT MASK	MOV SERPIN,C
JMP CNT	JMP CNT	
HI: ORI MASK	HI: ORI SERPRT,#MASK	
CNT: OUT SERPORT	CNT:	
RESULTS:		
10 INSTRUCTIONS	8 INSTRUCTIONS	4 INSTRUCTIONS
20 BYTES	13 BYTES	7 BYTES
72 STATES	11 CYCLES	5 CYCLES
24 uSEC.	27.5 uSEC.	5 uSEC.

Design Example #3—Combinatorial Logic Equations

Next we'll look at some simple uses for bit-test instructions and logical operations. (This example is also presented in Application Note AP-69.)

Virtually all hardware designers have solved complex functions using combinatorial logic. While the hardware involved may vary from relay logic, vacuum tubes, or TTL or to more esoteric technologies like fluidics, in each case the goal is the same: to solve a problem represented by a logical function of several Boolean variables.

Figure 13 shows TTL and relay logic diagrams for a function of the six variables U through Z. Each is a solution of the equation.

$$Q = (U \cdot (V + W)) + (X \cdot \bar{Y}) + \bar{Z}$$

Equations of this sort might be reduced using Karnaugh Maps or algebraic techniques, but that is not the purpose of this example. As the logic complexity increases, so does the difficulty of the reduction process. Even a minor change to the function equations as the design evolves would require tedious re-reduction from scratch.

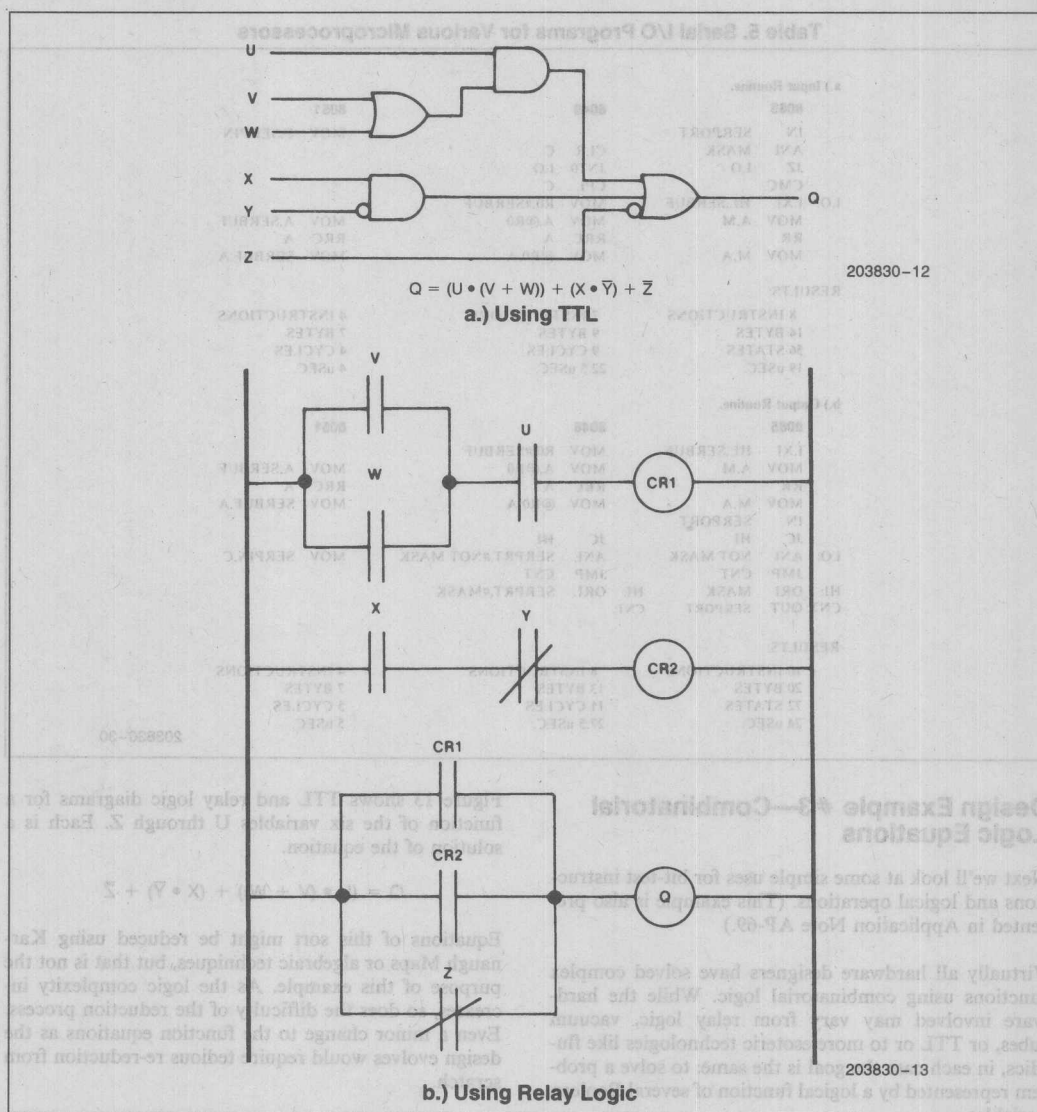


Figure 13. Hardware Implementations of Boolean Functions

For the sake of comparison we will implement this function three ways, restricting the software to three proper subsets of the MCS-51 instruction set. We will also assume that U and V are input pins from different input ports, W and X are status bits for two peripheral controllers, and Y and Z are software flags set up earlier in the program. The end result must be written

to an output pin on some third port. The first two implementations follow the flow-chart shown in Figure 14. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP—as soon as the proper result has been determined. These exits then rewrite the output port with the result bit respectively one or zero.

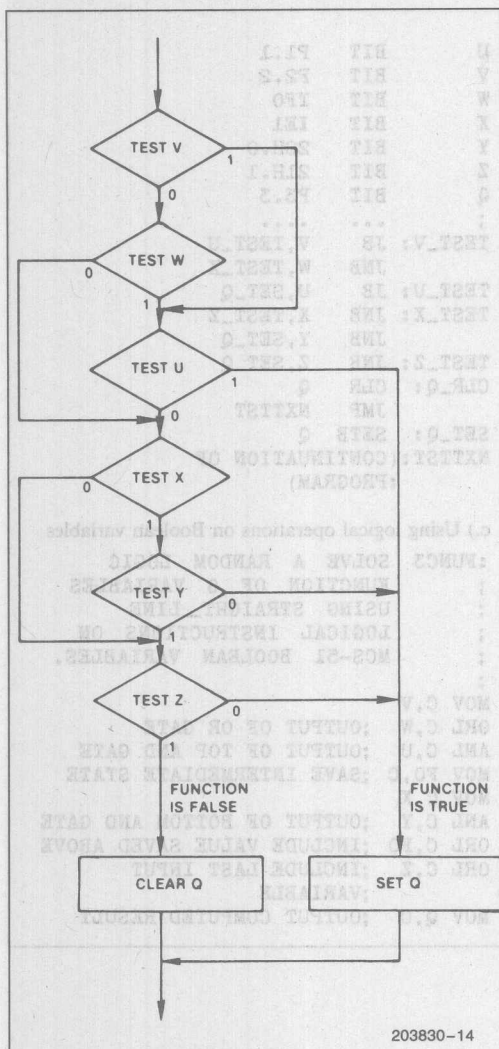


Figure 14. Flow Chart for Tree-Branching Algorithm

Other digital computers must solve equations of this type with standard word-wide logical instructions and conditional jumps. So for the first implementation, we won't use any generalized bit-addressing instructions. As we shall soon see, being constrained to such an instruction subset produces somewhat sloppy software solutions. MCS-51 mnemonics are used in Example 2a: other machines might further cloud the situation by requiring operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the MOV mnemonic used for all variable transfers in the 8051 instruction set.

The code which results is cumbersome and error prone. It would be difficult to prove whether the software worked for all input combinations in programs of this sort. Furthermore, execution time will vary widely with input data.

Thanks to the direct bit-test operations, a single instruction can replace each move mask conditional jump sequence in Example 2a, but the algorithm would be equally convoluted (see Example 2b). To lessen the confusion "a bit" each input variable is assigned a symbolic name.

A more elegant and efficient implementation (Example 2c) strings together the Boolean ANL and ORL functions to generate the output function with straight-line code. When finished, the carry flag contains the result, which is simply copied out to the destination pin. No flow chart is needed—code can be written directly from the logic diagrams in Figure 14. The result is simplicity itself: fast, flexible, reliable, easy to design, and easy to debug.

An 8051 program can simulate an N-input AND or OR gate with at most N + 1 lines of source program—one for each input and one line to store the results. To simulate NAND and NOR gates, complement the carry after computing the function. When some inputs to the gate have "inversion bubbles", perform the ANL or ORL operation on inverted operands. When the first input is inverted, either load the operand into the carry and then complement it, or use DeMorgan's Theorem to convert the gate to a different form.

Example 2. Software Solutions to Logic Function of Figure 13.

a.) Using only byte-wide logical instructions

```

:BFUNCI SOLVE RANDOM LOGIC
;
; FUNCTION OF 6 VARIABLES
; BY LOADING AND MASKING
; THE APPROPRIATE BITS IN
; THE ACCUMULATOR. THEN
; EXECUTING CONDITIONAL
; JUMPS BASED ON ZERO
; CONDITION. (APPROACH USED
; BY BYTE-ORIENTED
; ARCHITECTURES.) BYTE AND
; MASK VALUES CORRESPOND TO
; RESPECTIVE BYTE ADDRESS
; AND BIT POSITIONS.
;
;
OUTBUF DATA 22H
;OUTPUT PIN STATE MAP
;

```

```

;B1.F. MOV A,P1
ANL A,#00000100B
JNZ TESTU
MOV A,TCON
ANL A,#00100000B
JNZ TESTX
TESTU: MOV A,P1
ANL A,#000000010B
JNZ SETQ
TESTX: MOV A,TCON
ANL A,#00001000B
JZ TESTZ
MOV A,20H
ANL A,#00000001B
JZ SETQ
TESTZ: MOV A,21H
ANL A,#000000010B
JZ SETQ
CLRQ: MOV A,OUTBUF
ANL A,#11110111B
JMP OUTQ
SETQ: MOV A,OUTBUF
ORL A,#00001000B
OUTQ: MOV OUTBUF,A
MOV P3,A

```

b.) Using only bit-test instructions

```

:BFUNC2 SOLVE A RANDOM LOGIC
;
; FUNCTION OF 6 VARIABLES
; BY DIRECTLY POLLING EACH
; BIT. (APPROACH USING
; MCS-51 UNIQUE BIT-TEST
; INSTRUCTION CAPABILITY.)
; SYMBOLS USED IN LOGIC
; DIAGRAM ASSIGNED TO
; CORRESPONDING 8x51 BIT
; ADDRESSES.
;
;
;

```

```

U D11 P1.1
V BIT P2.2
W BIT TFO
X BIT IE1
Y BIT 20H.0
Z BIT 21H.1
Q BIT P3.3
;
;
TEST_V: JB V,TEST_U
JNB W,TEST_X
TEST_U: JB U,SET_Q
TEST_X: JNB X,TEST_Z
JNB Y,SET_Q
TEST_Z: JNB Z,SET_Q
CLR_Q: CLR Q
JMP NXTTST
SET_Q: SETB Q
NXTTST: (CONTINUATION OF
:PROGRAM)

```

c.) Using logical operations on Boolean variables

```

:FUNC3 SOLVE A RANDOM LOGIC
;
; FUNCTION OF 6 VARIABLES
; USING STRAIGHT_LINE
; LOGICAL INSTRUCTIONS ON
; MCS-51 BOOLEAN VARIABLES.
;
;
MOV C,V
ORL C,W ;OUTPUT OF OR GATE
ANL C,U ;OUTPUT OF TOP AND GATE
MOV FO,C ;SAVE INTERMEDIATE STATE
MOV C,X
ANL C,Y ;OUTPUT OF BOTTOM AND GATE
ORL C,FO ;INCLUDE VALUE SAVED ABOVE
ORL C,Z ;INCLUDE LAST INPUT
;VARIABLE
MOV Q,C ;OUTPUT COMPUTED RESULT

```

Figure 14 Flow Chart for Tree-Branching Algorithm

Other digital computers must solve equations of this type with standard word-wide logical instructions and conditional jumps. So for the first implementation, we won't use any generalized bit-addressing instructions. As we shall soon see, being constrained to such an instruction set produces somewhat idiosyncratic software solutions. MCS-51 microcontrollers are used in Example 2a; other machines might further cloud the situation by requiring operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the MOV mnemonic used for all variable transfers in the 8051 instruction set.

were to simulate a large number of gates by summing the total number of inputs and outputs. The *actual* total should be somewhat shorter, since calculations can be "chained," as shown. The output of one gate is often the first input to another, bypassing the intermediate variable to eliminate two lines of source.

Design Example #4—Automotive Dashboard Functions

Now let's apply these techniques to designing the software for a complete controller system. This application is patterned after a familiar real-world application which isn't nearly as trivial as it might first appear: automobile turn signals.

column as a single-pole, triple-throw toggle switch. In its central position all contacts are open. In the up or down positions contacts close causing corresponding lights in the rear of the car to blink. So far very simple.

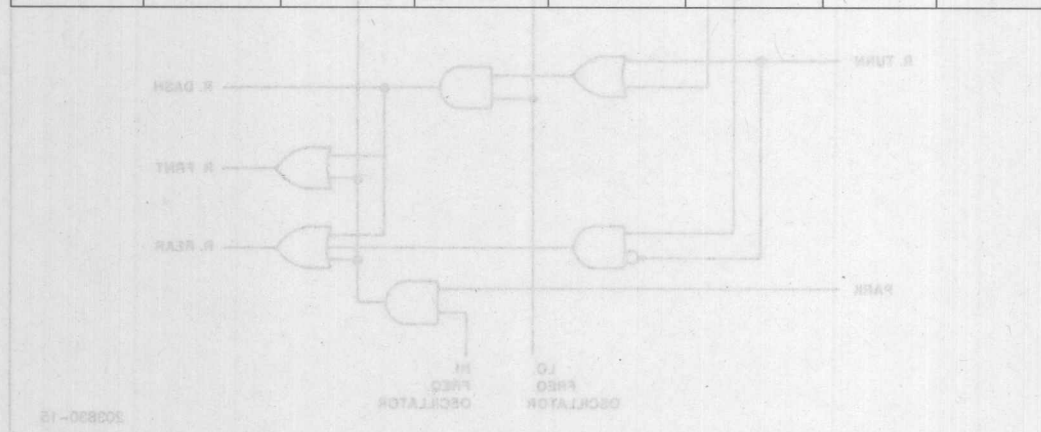
Two more turn signals blink in the front of the car, and two others in the dashboard. All six bulbs flash when an emergency switch is closed. A thermo-mechanical relay (accessible under the dashboard in case it wears out) causes the blinking.

Applying the brake pedal turns the tail light filaments on constantly . . . unless a turn is in progress, in which case the blinking tail light is not affected. (Of course, the front turn signals and dashboard indicators are not affected by the brake pedal.) Table 6 summarizes these operating modes.

2

Table 6. Truth Table for Turn-Signal Operation

Input Signals				Output Signals			
Brake Switch	Emerg. Switch	Left Turn Switch	Right Turn Switch	Left Front & Dash	Right Front & Dash	Left Rear	Right Rear
0	0	0	0	Off	Off	Off	Off
0	0	0	1	Off	Blink	Off	Blink
0	0	1	0	Blink	Off	Blink	Off
0	1	0	0	Blink	Blink	Blink	Blink
0	1	0	1	Blink	Blink	Blink	Blink
0	1	1	0	Blink	Blink	Blink	Blink
1	0	0	0	Off	Off	On	On
1	0	0	1	Off	Blink	On	Blink
1	0	1	0	Blink	Off	Blink	On
1	1	0	0	Blink	Blink	On	On
1	1	0	1	Blink	Blink	On	Blink
1	1	1	0	Blink	Blink	Blink	On



But we're not done yet. Each of the exterior turn signal (but not the dashboard) bulbs has a second, somewhat dimmer filament for the parking lights. Figure 15 shows TTL circuitry which could control all six bulbs. The signals labeled "High Freq." and "Low Freq." represent two square-wave inputs. Basically, when one of the turn switches is closed or the emergency switch is activated the low frequency signal (about 1 Hz) is gated through to the appropriate dashboard indicator(s) and turn signal(s). The rear signals are also activated when the brake pedal is depressed provided a turn is not being made in the same direction. When the parking light switch is closed the higher frequency oscillator is gated to each front and rear turn signal, sustaining a low-intensity background level. (This is to eliminate the need for additional parking light filaments.)

In most cars, the switching logic to generate these functions requires a number of multiple-throw contacts. As many as 18 conductors thread the steering column of some automobiles solely for turn-signal and emergency blinker functions. (The author discovered this recently to his astonishment and dismay when replacing the whole assembly because of one burned contact.)

A multiple-conductor wiring harness runs to each corner of the car, behind the dash, up the steering column, and down to the blinker relay below. Connectors at

each termination for each filament lead to extra cost and labor during construction, lower reliability and safety, and more costly repairs. And considering the system's present complexity, increasing its reliability or detecting failures would be quite difficult.

There are two reasons for going into such painful detail describing this example. First, to show that the messiest part of many system designs is determining what the controller should do. Writing the software to solve these functions will be comparatively easy. Secondly, to show the many potential failure points in the system. Later we'll see how the peripheral functions and intelligence built into a microcomputer (with a little creativity) can greatly reduce external interconnections and mechanical part count.

The Single-Chip Solution

The circuit shown in Figure 16 indicates five input pins to the five input variables—left-turn select, right-turn select, brake pedal down, emergency switch on, and parking lights on. Six output pins turn on the front, rear, and dashboard indicators for each side. The microcomputer implements all logical functions through software, which periodically updates the output signals as time elapses and input conditions change.

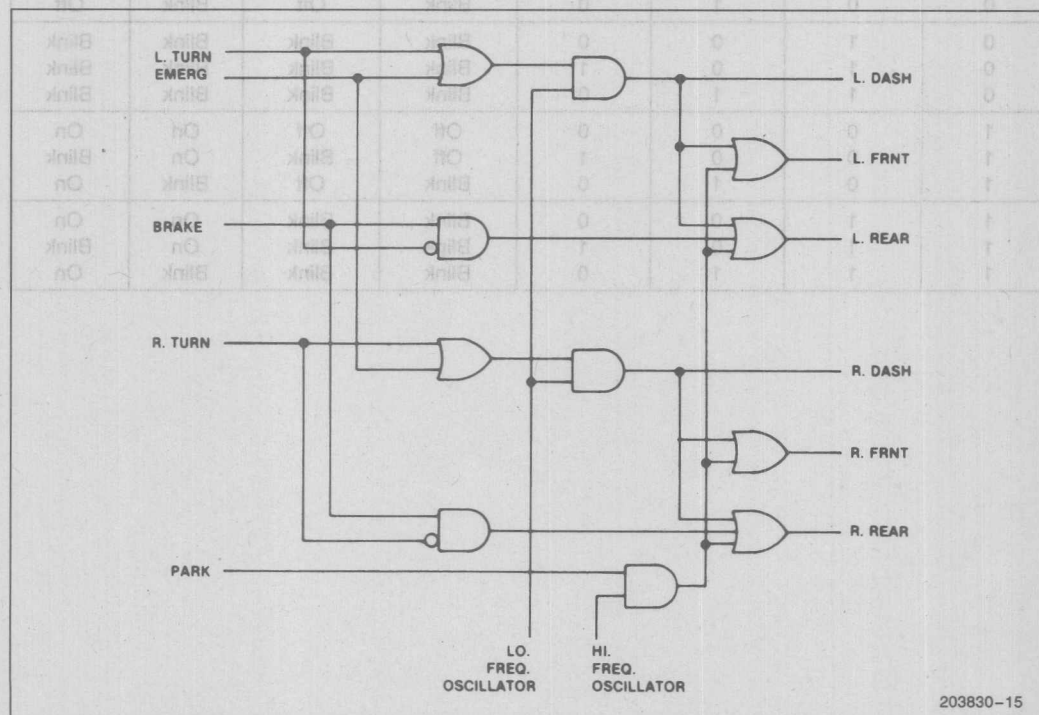


Figure 15. TTL Logic Implementation of Automotive Turn Signals

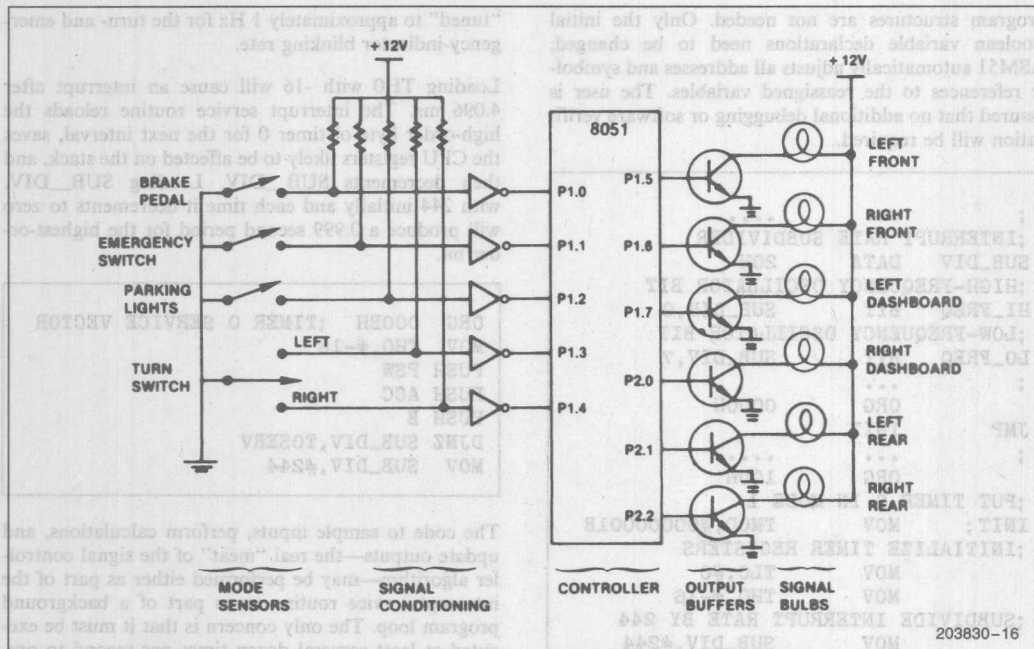


Figure 16. Microcomputer Turn-Signal Connections

Design Example #3 demonstrated that symbolic addressing with user-defined bit names makes code and documentation easier to write and maintain. Accordingly, we'll assign these I/O pins names for use throughout the program. (The format of this example will differ somewhat from the others. Segments of the overall program will be presented in sequence as each is described.)

```

;
; INPUT PIN DECLARATIONS:
; (ALL INPUTS ARE POSITIVE-TRUE LOGIC)
;
BRAKE BIT P1.0 ;BRAKE PEDAL
;DEPRESSED
EMERG BIT P1.1 ;EMERGENCY BLINKER
;ACTIVATED
PARK BIT P1.2 ;PARKING LIGHTS ON
I_TURN BIT P1.3 ;TURN LEVER DOWN
R_TURN BIT P1.4 ;TURN LEVER UP
;
; OUTPUT PIN DECLARATIONS:
;
I_FRNT BIT P1.5 ;FRONT LEFT-TURN
;INDICATOR
R_FRNT BIT P1.6 ;FRONT RIGHT-TURN
;INDICATOR
I_DASH BIT P1.7 ;DASHBOARD LEFT-TURN
;INDICATOR

```

```

R_DASH BIT P2.0 ;DASHBOARD RIGHT-
;TURN INDICATOR
I_REAR BIT P2.1 ;REAR LEFT-TURN
;INDICATOR
R_REAR BIT P2.2 ;REAR RIGHT-TURN
;INDICATOR
;

```

Another key advantage of symbolic addressing will appear further on in the design cycle. The locations of cable connectors, signal conditioning circuitry, voltage regulators, heat sinks, and the like all affect P.C. board layout. It's quite likely that the somewhat arbitrary pin assignment defined early in the software design cycle will prove to be less than optimum; rearranging the I/O pin assignment could well allow a more compact module, or eliminate costly jumpers on a single-sided board. (These considerations apply especially to automotive and other cost-sensitive applications needing single-chip controllers.) Since other architectures mask bytes or use "clever" algorithms to isolate bits by rotating them into the carry, re-routing an input signal (from bit 1 of port 1, for example, to bit 4 of port 3) could require extensive modifications throughout the software.

The Boolean Processor's direct bit addressing makes such changes absolutely trivial. The number of the port containing the pin is irrelevant, and masks and complex

program structures are not needed. Only the initial Boolean variable declarations need to be changed; ASM51 automatically adjusts all addresses and symbolic references to the reassigned variables. The user is assured that no additional debugging or software verification will be required.

```

;      ...      .....
;INTERUPT RATE SUBDIVIDER
SUB_DIV DATA 20H
;HIGH-FREQUENCY OSCILLATOR BIT
HI_FREQ BIT SUB_DIV,0
;LOW-FREQUENCY OSCILLATOR BIT
LO_FREQ BIT SUB_DIV,7
;
;      ...      .....
;      ORG 0000H
JMP INIT
;
;      ...      .....
;      ORG 100H
;PUT TIMER 0 IN MODE 1
INIT; MOV TMOD,#00000001B
;INITIALIZE TIMER REGISTERS
MOV TLO,#0
MOV TH0,#-16
;SUBDIVIDE INTERRUPT RATE BY 244
MOV SUB_DIV,#244
;ENABLE TIMER INTERRUPTS
SETB ETO
;GLOBALLY ENABLE ALL INTERRUPTS
SETB EA
;START TIMER
SETB TRO
;
; (CONTINUE WITH BACKGROUND PROGRAM)
;
;PUT TIMER 0 IN MODE 1
;INITIALIZE TIMER REGISTERS

;SUBDIVIDE INTERRUPT RATE BY 244
;ENABLE TIMER INTERRUPTS
;GLOBALLY ENABLE ALL INTERRUPTS
;START TIMER

```

Timer 0 (one of the two on-chip timer counters) replaces the thermo-mechanical blinker relay in the dash-board controller. During system initialization it is configured as a timer in mode 1 by setting the least significant bit of the timer mode register (TMOD). In this configuration the low-order byte (TLO) is incremented every machine cycle, overflowing and incrementing the high-order byte (TH0) every 256 μ s. Timer interrupt 0 is enabled so that a hardware interrupt will occur each time TH0 overflows.

An eight-bit variable in the bit-addressable RAM array will be needed to further subdivide the interrupts via software. The lowest-order bit of this counter toggles very fast to modulate the parking lights: bit 7 will be

"tuned" to approximately 1 Hz for the turn- and emergency-indicator blinking rate.

Loading TH0 with -16 will cause an interrupt after 4.096 ms. The interrupt service routine reloads the high-order byte of timer 0 for the next interval, saves the CPU registers likely to be affected on the stack, and then decrements SUB_DIV. Loading SUB_DIV with 244 initially and each time it decrements to zero will produce a 0.999 second period for the highest-order bit.

```

ORG 000BH ;TIMER 0 SERVICE VECTOR
MOV TH0,#-16
PUSH PSW
PUSH ACC
PUSH B
DJNZ SUB_DIV,TOSERV
MOV SUB_DIV,#244

```

The code to sample inputs, perform calculations, and update outputs—the real "meat" of the signal controller algorithm—may be performed either as part of the interrupt service routine or as part of a background program loop. The only concern is that it must be executed at least several dozen times per second to prevent parking light flickering. We will assume the former case, and insert the code into the timer 0 service routine.

First, notice from the logic diagram (Figure 15) that the subterm (PARK • H_FREQ), asserted when the parking lights are to be on dimly, figures into four of the six output functions. Accordingly, we will first compute that term and save it in a temporary location named "DIM". The PSW contains two general purpose flags: F0, which corresponds to the 8048 flag of the same name, and PSW.1. Since the PSW has been saved and will be restored to its previous state after servicing the interrupt, we can use either bit for temporary storage.

```

DIM BIT PSW.1 ;DECLARE TEMP
;STORAGE FLAG
; ... .....
MOV C,PARK ;GATE PARKING
;LIGHT SWITCH
ANL HI_FREQ ;WITH HIGH
;FREQUENCY
;SIGNAL
MOV DIM,C ;AND SAVE IN
;TEMP. VARIABLE

```

This simple three-line section of code illustrates a remarkable point. The software indicates in very abstract terms exactly what function is being performed, inde-

pendent of the hardware configuration. The fact that these three bits include an input pin, a bit within a program variable, and a software flag in the PSW is totally invisible to the programmer.

Now generate and output the dashboard left turn signal.

```

;
MOV C,L_TURN ;SET CARRY IF
;TURN
ORL C,EMERG ;OR EMERGENCY
;SELECTED
ANL C,LO_FREQ ;GATE IN 1 HZ
;SIGNAL
MOV I_DASH,C ;AND OUTPUT TO
;DASHBOARD

```

To generate the left front turn signal we only need to add the parking light function in FO. But notice that the function in the carry will also be needed for the rear signal. We can save effort later by saving its current state in FO.

```

;
MOV FO,C ;SAVE FUNCTION
;SO FAR
ORL C,DIM ;ADD IN PARKING
;LIGHT FUNCTION
MOV L_FRNT,C ;AND OUTPUT TO
;TURN SIGNAL

```

Finally, the rear left turn signal should also be on when the brake pedal is depressed, provided a left turn is not in progress.

```

MOV C,BRAKE ;GATE BRAKE
;PEDAL SWITCH
ANL C,L_TURN ;WITH TURN
;LEVER
ORL C,FO ;INCLUDE TEMP.
;VARIABLE FROM DASH

```

```

ORL C,DIM ;AND PARKING
;LIGHT FUNCTION
MOV L_REAR,C ;AND OUTPUT TO
;TURN SIGNAL

```

Now we have to go through a similar sequence for the right-hand equivalents to all the left-turn lights. This also gives us a chance to see how the code segments above look when combined.

```

MOV C,R_TURN ;SET CARRY H-
;TURN
ORL C,EMERG ;OR EMERGENCY
;SELECTED
ANL C,LO_FREQ ;IF SO. GATE IN 1
;HZ SIGNAL
MOV R_DASH,C ;AND OUTPUT TO
;DASHBOARD
MOV FO,C ;SAVE FUNCTION
;SO FAR
ORL C,DIM ;ADD IN PARKING
;LIGHT FUNCTION
MOV R_FRNT,C ;AND OUTPUT TO
;TURN SIGNAL
MOV C,BRAKE ;GATE BRAKE
;PEDAL SWITCH
ANL C,R_TURN ;WITH TURN
;LEVER
ORL C,FO ;INCLUDE TEMP.
;VARIABLE FROM
;DASH
ORL C,DIM ;AND PARKING
;LIGHT FUNCTION
MOV R_REAR,C ;AND OUTPUT TO
;TURN SIGNAL

```

(The perceptive reader may notice that simply rearranging the steps could eliminate one instruction from each sequence.)

Now that all six bulbs are in the proper states, we can return from the interrupt routine, and the program is finished. This code essentially needs to reverse the status saving steps at the beginning of the interrupt.

Table 7. Non-Trivial Duty Cycles

Sub_Div Bits									Duty Cycles						
7	6	5	4	3	2	1	0		12.5%	25.0%	37.5%	50.0%	62.5%	75.0%	87.5%
X	X	X	X	X	0	0	0		Off	Off	Off	Off	Off	Off	Off
X	X	X	X	X	0	0	1		Off	Off	Off	Off	Off	Off	On
X	X	X	X	X	0	1	0		Off	Off	Off	Off	Off	On	On
X	X	X	X	X	0	1	1		Off	Off	Off	Off	On	On	On
X	X	X	X	X	1	0	0		Off	Off	Off	On	On	On	On
X	X	X	X	X	1	0	1		Off	Off	On	On	On	On	On
X	X	X	X	X	1	1	0		Off	On	On	On	On	On	On
X	X	X	X	X	1	1	1		On	On	On	On	On	On	On

```

POP B      ;RESTORE CPU
            ;REGISTERS.
POP ACC
POP PSW
RETI

```

Program Refinements. The luminescence of an incandescent light bulb filament is generally non-linear: the 50% duty cycle of HI_FREQ may not produce the desired intensity. If the application requires, duty cycles of 25%, 75%, etc. are easily achieved by ANDing and ORing in additional low-order bits of SUB_DIV. For example, 30 H/ signals of seven different duty cycles could be produced by considering bits 2-0 as shown in Table 7. The only software change required would be to the code which sets-up variable DIM;

```

MOV C,SUB_DIV.1;START WITH 50
                ;PERCENT
ANL C,SUB_DIV.0;MASK DOWN TO 25
                ;PERCENT
ORL C,SUB_DIV.2;AND BUILD BACK TO
                ;62 PERCENT
MOV DIM,C      ;DUTY CYCLE FOR
                ;PARKING LIGHTS.

```

Interconnections increase cost and decrease reliability. The simple buffered pin-per-function circuit in Figure 16 is insufficient when many outputs require higher-than-TTL drive levels. A lower-cost solution uses the 8051 serial port in the shift-register mode to augment I/O. In mode 0, writing a byte to the serial port data buffer (SBUF) causes the data to be output sequentially through the "RXD" pin while a burst of eight clock pulses is generated on the "TXD" pin. A shift register connected to these pins (Figure 17) will load the data byte as it is shifted out. A number of special peripheral

driver circuits combining shift-register inputs with high drive level outputs have been introduced recently.

Cascading multiple shift registers end-to-end will expand the number of outputs even further. The data rate in the I/O expansion mode is one megabaud, or 8 μ s. per byte. This is the mode which the serial port defaults to following a reset, so no initialization is required.

The software for this technique uses the B register as a "map" corresponding to the different output functions. The program manipulates these bits instead of the output pins. After all functions have been calculated the B register is shifted by the serial port to the shift-register driver. (While some outputs may glitch as data is shifted through them, at 1 Megabaud most people wouldn't notice. Some shift registers provide an "enable" bit to hold the output states while new data is being shifted in.)

This is where the earlier decision to address bits symbolically throughout the program is going to pay off. This major I/O restructuring is nearly as simple to implement as rearranging the input pins. Again, only the bit declarations need to be changed.

```

I_FRNT BIT B.0;FRONT LEFT-TURN
                ;INDICATOR
R_FRNT BIT B.1;FRONT RIGHT-TURN
                ;INDICATOR
I_DASH BIT B.2;DASHBOARD LEFT-TURN
                ;INDICATOR
R_DASH BIT B.3;DASHBOARD RIGHT-TURN
                ;INDICATOR
I_REAR BIT B.4;REAR LEFT-TURN
                ;INDICATOR
R_REAR BIT B.5;REAR RIGHT-TURN
                ;INDICATOR

```

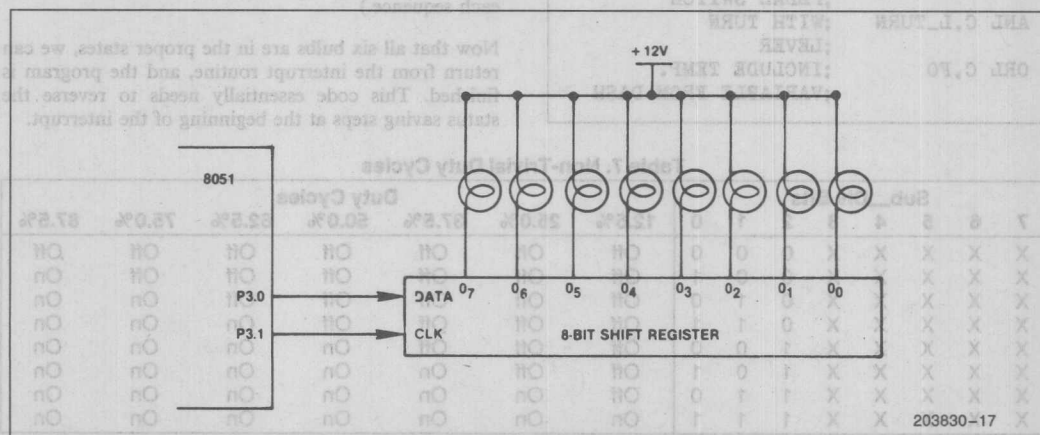


Figure 17. Output Expansion Using Serial Port

The original program to compute the functions need not change. After computing the output variables, the control map is transmitted to the buffered shift register through the serial port.

```
MOV SBUF,B ;LOAD BUFFER AND TRANSMIT
```

The Boolean Processor solution holds a number of advantages over older methods. Fewer switches are required. Each is simpler, requiring fewer poles and lower current contacts. The flasher relay is eliminated entirely. Only six filaments are driven, rather than 10. The wiring harness is therefore simpler and less expensive—one conductor for each of the six lamps and each of the five sensor switches. The fewer conductors use far fewer connectors. The whole system is more reliable.

And since the system is much simpler it would be feasible to implement redundancy and or fault detection on the four main turn indicators. Each could still be a

standard double filament bulb, but with the filaments driven in parallel to tolerate single-element failures.

Even with redundancy, the lights will eventually fail. To handle this inescapable fact current or voltage sensing circuits on each main drive wire can verify that each bulb and its high-current driver is functioning properly. Figure 18 shows one such circuit.

Assume all of the lights are turned on except one: i.e., all but one of the collectors are grounded. For the bulb which is turned off, if there is continuity from +12V through the bulb base and filament, the control wire, all connectors, and the P.C. board traces, and if the transistor is indeed not shorted to ground, then the collector will be pulled to +12V. This turns on the base of Q8 through the corresponding resistor, and grounds the input pin, verifying that the bulb circuit is operational. The continuity of each circuit can be checked by software in this way.

2

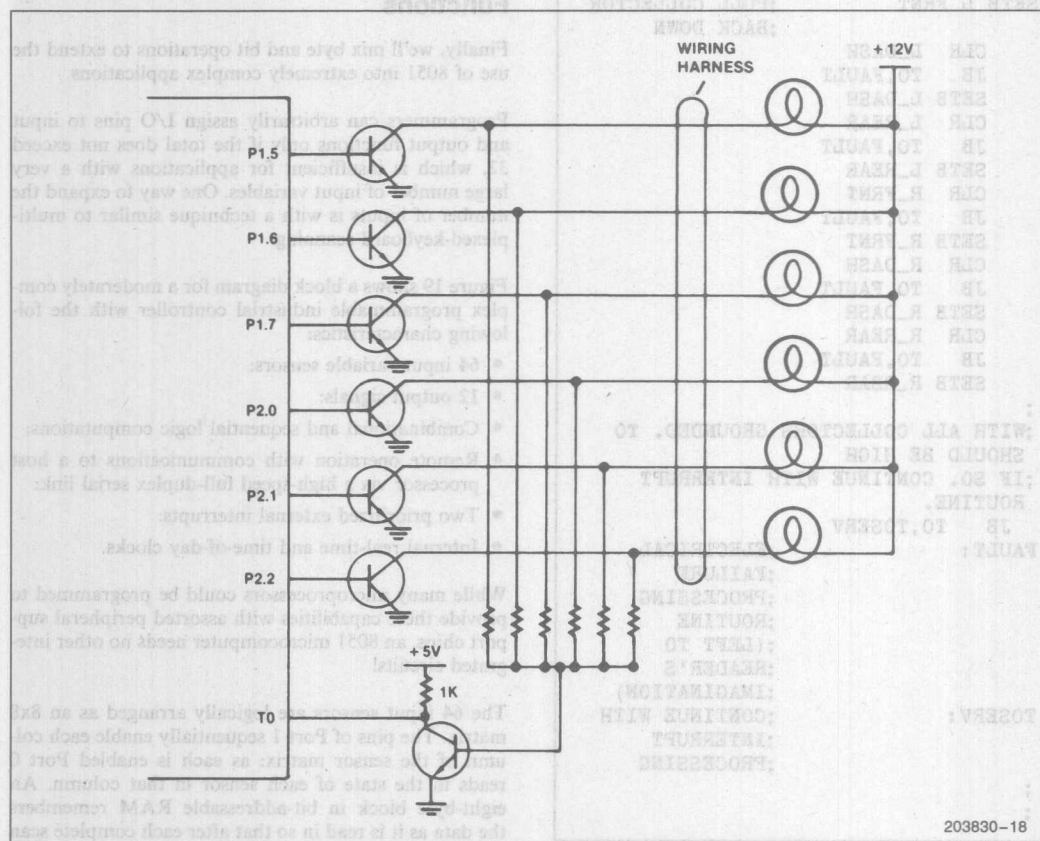


Figure 18

Now turn *all* the bulbs on, grounding all the collectors. Q7 should be turned off, and the Test pin should be high. However, a control wire shorted to +12V or an open-circuited drive transistor would leave one of the collectors at the higher voltage even now. This too would turn on Q7, indicating a different type of failure. Software could perform these checks once per second by executing the routine every time the software counter SUB_DIV is reloaded by the interrupt routine.

```

DINZ SUB_DIV,TOSERV
MOV SUB_DIV,#244      ;RELOAD COUNTER
ORL P1,#11100000B    ;SET CONTROL
                      ;OUTPUTS HIGH
ORL P2,#00000111B
CLR I_FRNT            ;FLOAT DRIVE
                      ;COLLECTOR
JB TO,FAULT          ;TO SHOULD BE
                      ;PULLED LOW
SETB L_FRNT           ;PULL COLLECTOR
                      ;BACK DOWN

CLR L_DASH
JB TO,FAULT
SETB L_DASH
CLR L_REAR
JB TO,FAULT
SETB L_REAR
CLR R_FRNT
JB TO,FAULT
SETB R_FRNT
CLR R_DASH
JB TO,FAULT
SETB R_DASH
CLR R_REAR
JB TO,FAULT
SETB R_REAR

;
;WITH ALL COLLECTORS GROUNDED. TO
;SHOULD BE HIGH
;IF SO. CONTINUE WITH INTERRUPT
;ROUTINE.
JB TO,TOSERV
FAULT:                ;ELECTRICAL
                      ;FAILURE
                      ;PROCESSING
                      ;ROUTINE
                      ;(LEFT TO
                      ;READER'S
                      ;IMAGINATION)
TOSERV:                ;CONTINUE WITH
                      ;INTERRUPT
                      ;PROCESSING
;
;
;

```

The complete assembled program listing is printed in Appendix A. The resulting code consists of 67 program statements, not counting declarations and comments, which assemble into 150 bytes of object code. Each pass through the service routine requires (coincidentally) 67 μ s plus 32 μ s once per second for the electrical test. If executed every 4 ms as suggested this software would typically reduce the throughput of the background program by less than 2%.

Once a microcomputer has been designed into a system, new features suddenly become virtually free. Software could make the emergency blinkers flash alternately or at a rate faster than the turn signals. Turn signals could override the emergency blinkers. Adding more bulbs would allow multiple tail light sequencing and synchopation—true flash factor, so to speak.

Design Example #5—Complex Control Functions

Finally, we'll mix byte and bit operations to extend the use of 8051 into extremely complex applications.

Programmers can arbitrarily assign I/O pins to input and output functions only if the total does not exceed 32, which is insufficient for applications with a very large number of input variables. One way to expand the number of inputs is with a technique similar to multiplexed-keyboard scanning.

Figure 19 shows a block diagram for a moderately complex programmable industrial controller with the following characteristics:

- 64 input variable sensors:
- 12 output signals:
- Combinational and sequential logic computations:
- Remote operation with communications to a host processor via a high-speed full-duplex serial link:
- Two prioritized external interrupts:
- Internal real-time and time-of-day clocks.

While many microprocessors could be programmed to provide these capabilities with assorted peripheral support chips, an 8051 microcomputer needs no other integrated circuits!

The 64 input sensors are logically arranged as an 8x8 matrix. The pins of Port 1 sequentially enable each column of the sensor matrix: as each is enabled Port 0 reads in the state of each sensor in that column. An eight-byte block in bit-addressable RAM remembers the data as it is read in so that after each complete scan cycle there is an internal map of the current state of all sensors. Logic functions can then directly address the elements of the bit map.

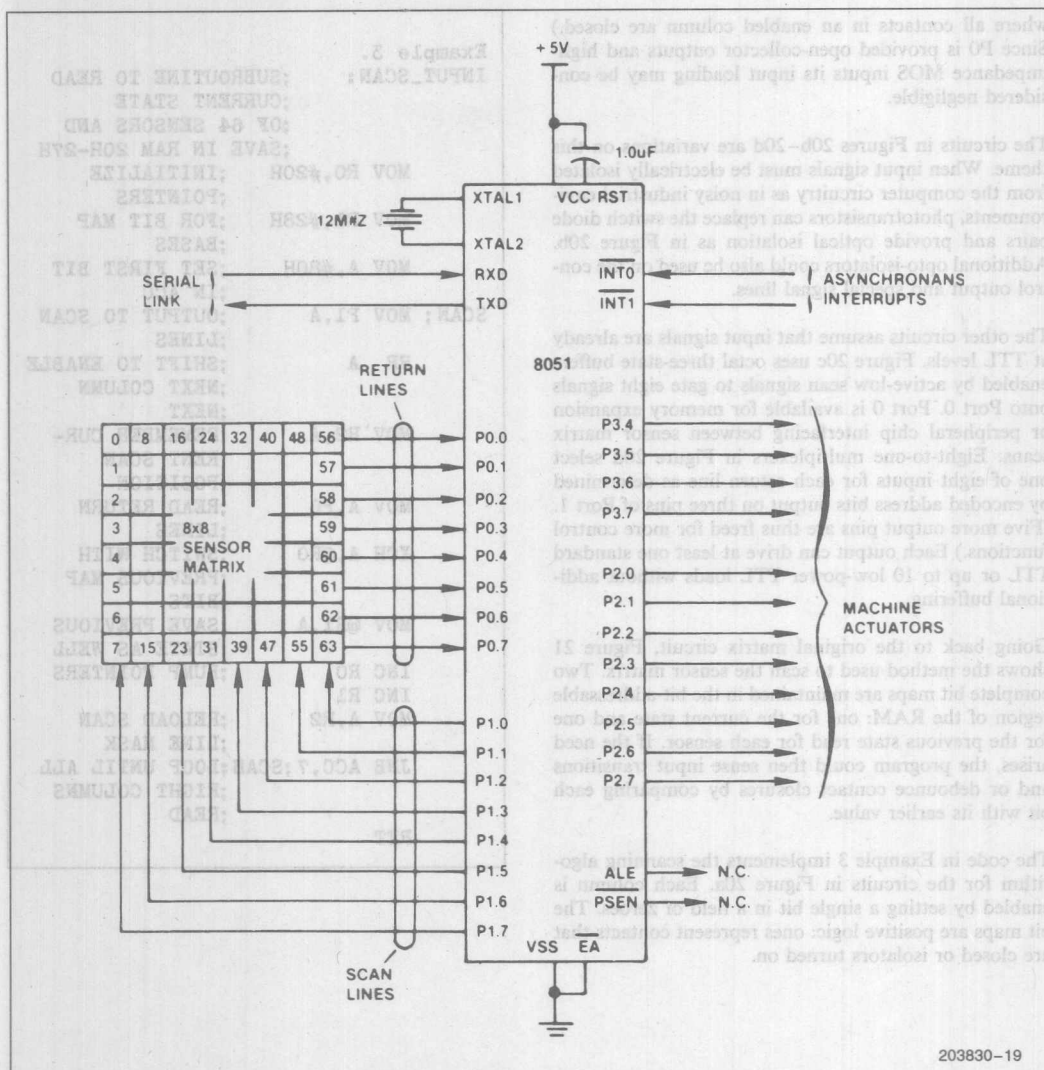


Figure 19. Block Diagram of 64-Input Machine Controller

The computer's serial port is configured as a nine-bit UART, transferring data at 17,000 bytes-per-second. The ninth bit may distinguish between address and data bytes.

The 8051 serial port can be configured to detect bytes with the address bit set, automatically ignoring all others. Pins INT0 and INT1 are interrupts configured respectively as high-priority, falling-edge triggered and low-priority, low-level triggered. The remaining 12 I/O pins output TTL-level control signals to 12 actuators.

There are several ways to implement the sensor matrix circuitry, all logically similar. Figure 20a shows one possibility. Each of the 64 sensors consists of a pair of simple switch contacts in series with a diode to permit multiple contact closures throughout the matrix.

The scan lines from Port 1 provide eight un-encoded active-high scan signals for enabling columns of the matrix. The return lines on rows where a contact is closed are pulled high and read as logic ones. Open return lines are pulled to ground by one of the 40 k Ω resistors and are read as zeroes. (The resistor values must be chosen to ensure all return lines are pulled above the 2.0V logic threshold, even in the worst-case,

where all contacts in an enabled column are closed.) Since P0 is provided open-collector outputs and high-impedance MOS inputs its input loading may be considered negligible.

The circuits in Figures 20b-20d are variations on this theme. When input signals must be electrically isolated from the computer circuitry as in noisy industrial environments, phototransistors can replace the switch diode pairs and provide optical isolation as in Figure 20b. Additional opto-isolators could also be used on the control output and special signal lines.

The other circuits assume that input signals are already at TTL levels. Figure 20c uses octal three-state buffers enabled by active-low scan signals to gate eight signals onto Port 0. Port 0 is available for memory expansion or peripheral chip interfacing between sensor matrix scans. Eight-to-one multiplexers in Figure 20d select one of eight inputs for each return line as determined by encoded address bits output on three pins of Port 1. (Five more output pins are thus freed for more control functions.) Each output can drive at least one standard TTL or up to 10 low-power TTL loads without additional buffering.

Going back to the original matrix circuit, Figure 21 shows the method used to scan the sensor matrix. Two complete bit maps are maintained in the bit-addressable region of the RAM: one for the current state and one for the previous state read for each sensor. If the need arises, the program could then sense input transitions and or debounce contact closures by comparing each bit with its earlier value.

The code in Example 3 implements the scanning algorithm for the circuits in Figure 20a. Each column is enabled by setting a single bit in a field of zeroes. The bit maps are positive logic: ones represent contacts that are closed or isolators turned on.

Example 3.

```

INPUT_SCAN:      ;SUBROUTINE TO READ
                  ;CURRENT STATE
                  ;OF 64 SENSORS AND
                  ;SAVE IN RAM 20H-27H

MOV R0,#20H      ;INITIALIZE
                  ;POINTERS
MOV R1,#28H      ;FOR BIT MAP
                  ;BASES
MOV A,#80H       ;SET FIRST BIT
                  ;IN ACC
SCAN: MOV P1,A    ;OUTPUT TO SCAN
                  ;LINES
RR A             ;SHIFT TO ENABLE
                  ;NEXT COLUMN
                  ;NEXT
MOV R2,A         ;REMEMBER CUR-
                  ;RENT SCAN
MOV A,P0         ;POSITION
                  ;READ RETURN
XCH A,@R0        ;LINES
                  ;SWITCH WITH
MOV @R1,A        ;PREVIOUS MAP
                  ;BITS
INC R0           ;SAVE PREVIOUS
INC R1           ;STATE AS WELL
MOV A,R2         ;BUMP POINTERS
JNB ACC,7;SCAN  ;RELOAD SCAN
                  ;LINE MASK
                  ;LOOP UNTIL ALL
                  ;EIGHT COLUMNS
                  ;READ
RET
    
```

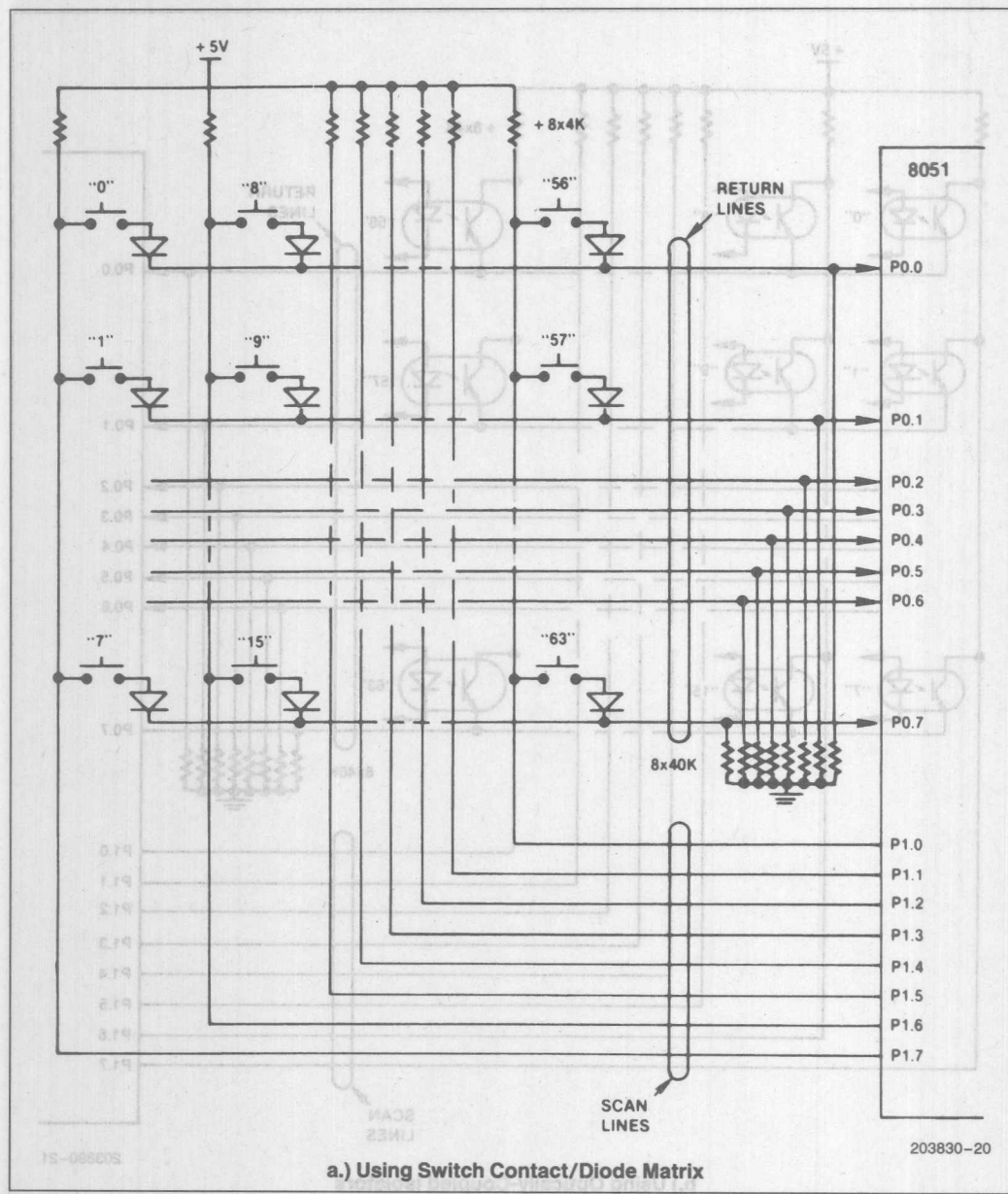


Figure 20. Sensor Matrix Implementation Methods

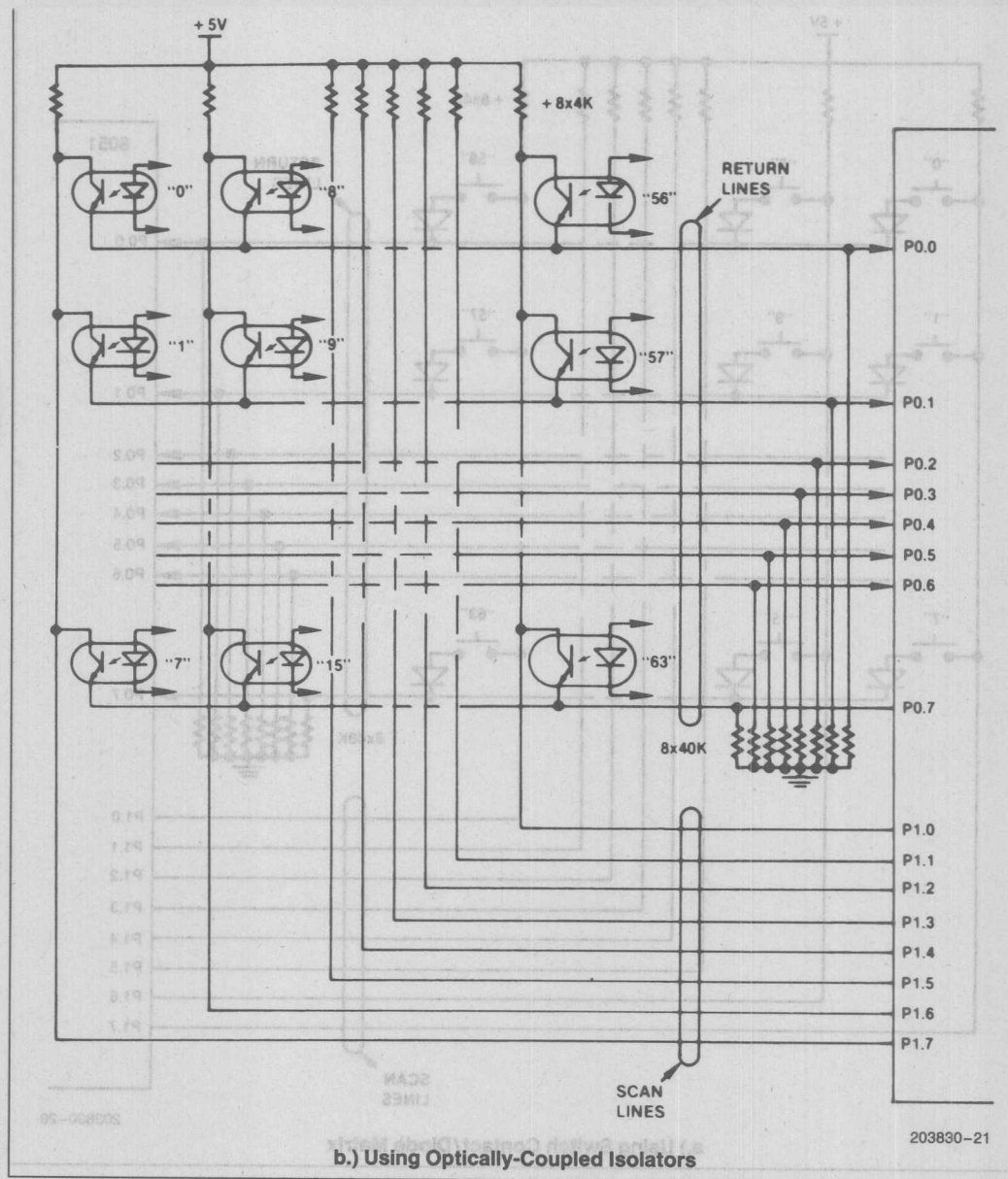
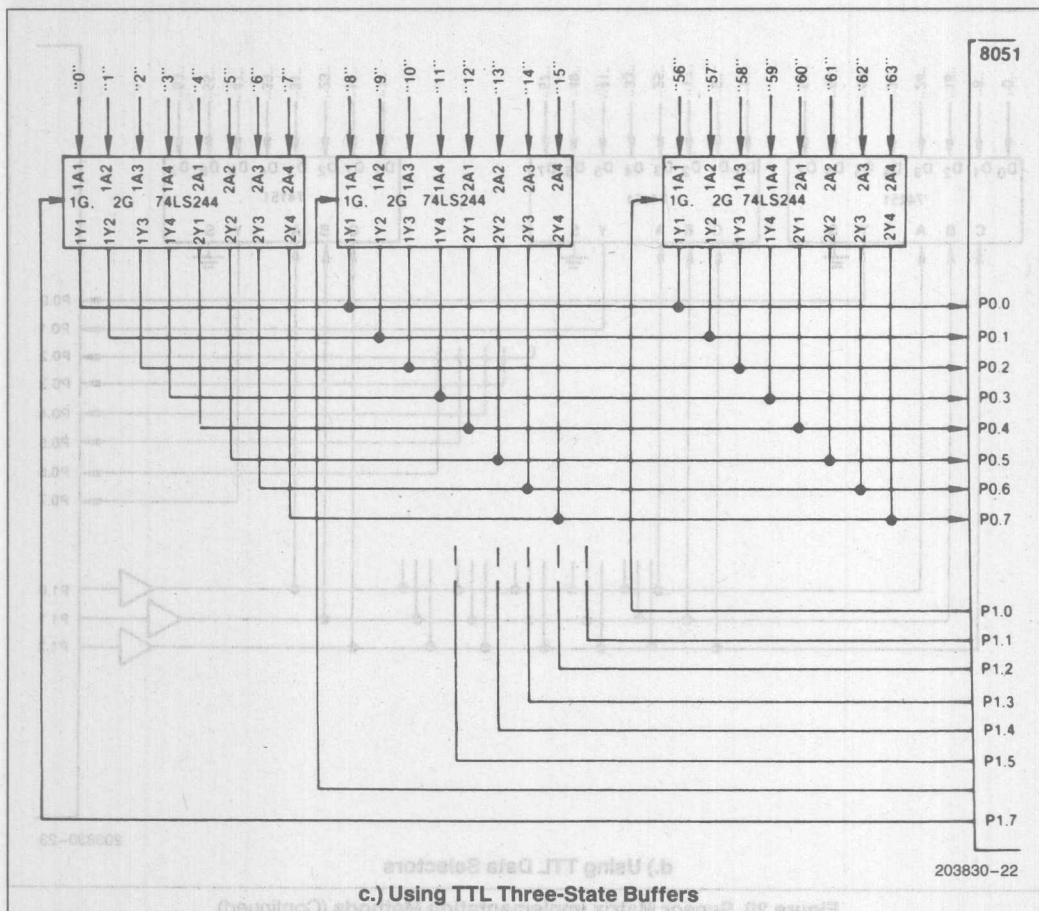


Figure 20. Sensor Matrix Implementation Methods (Continued)



c.) Using TTL Three-State Buffers

Figure 20. Sensor Matrix Implementation Methods (Continued)

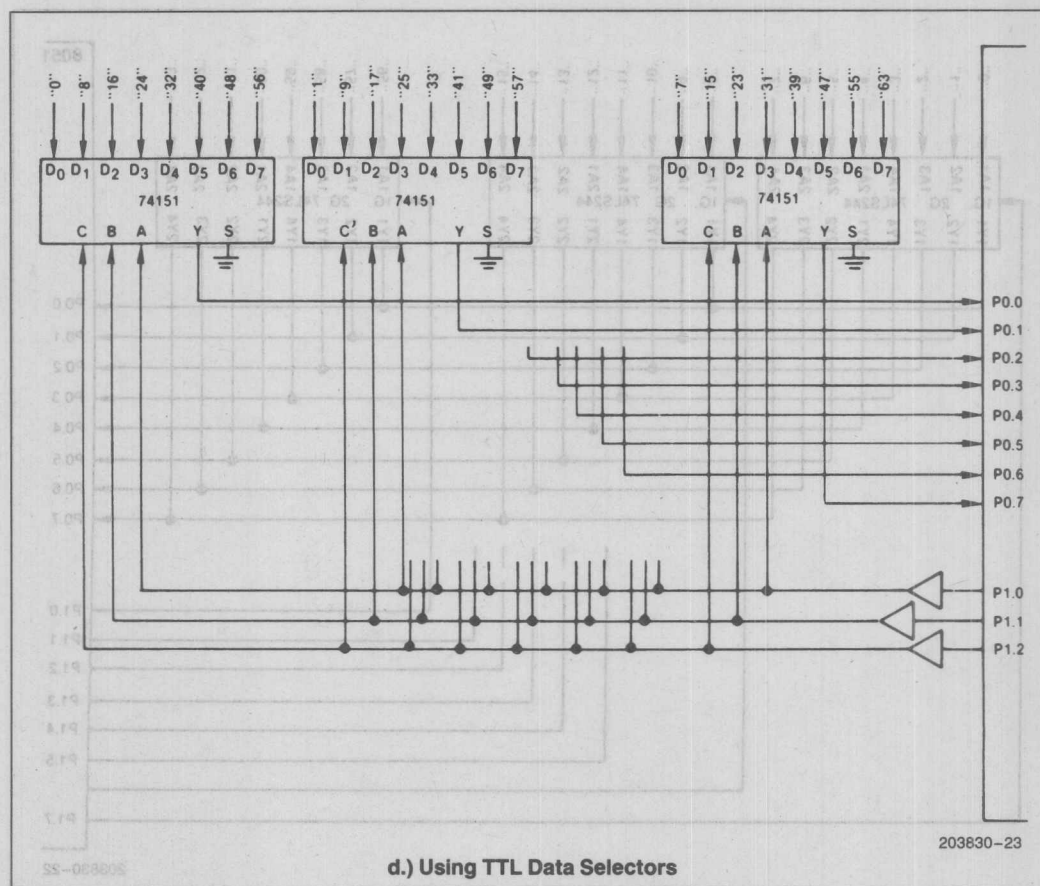


Figure 20. Sensor Matrix Implementation Methods (Continued)

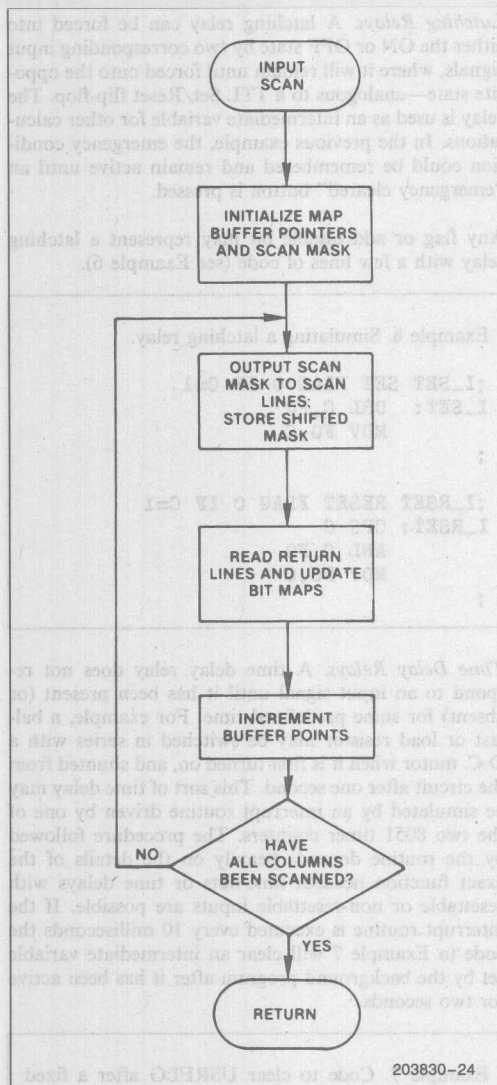


Figure 21. Flowchart for Reading in Sensor Matrix

What happens after the sensors have been scanned depends on the individual application. Rather than in-

venting some artificial design problem, software corresponding to commonplace logic elements will be discussed.

Combinatorial Output Variables. An output variable which is a simple (or not so simple) combinational function of several input variables is computed in the spirit of Design Example 3. All 64 inputs are represented in the bit maps: in fact, the sensor numbers in Figure 20 correspond to the absolute bit addresses in RAM! The code in Example 4 activates an actuator connected to P2.2 when sensors 12, 23, and 34 are closed and sensors 45 and 56 are open.

Example 4.

Simple Combinatorial Output Variables.

```

;SET P2.2=(12)(23)(34)(45)(56)
MOV C,12
ANL C,23
ANL C,34
ANL C,45
ANL C,56
MOV P2.2,C
  
```

Intermediate Variables. The examination of a typical relay-logic ladder diagram will show that many of the rungs control *not* outputs but rather relays whose contacts figure into the computation of other functions. In effect, these relays indicate the state of intermediate variables of a computation.

The MCS-51 solution can use any directly addressable bit for the storage of such intermediate variables. Even when all 128 bits of the RAM array are dedicated (to input bit maps in this example), the accumulator, PSW, and B register provide 18 additional flags for intermediate variables.

For example, suppose switches 0 through 3 control a safety interlock system. Closing any of them should deactivate certain outputs. Figure 22 is a ladder diagram for this situation. The interlock function could be re-computed for every output affected, or it may be computed once and save (as implied by the diagram). As the program proceeds this bit can qualify each output.

Example 5. Incorporating Override signal into actuator outputs.

```

;      CALL INPUT_SCAN
      MOV C,0
      ORL C,1
      ORL C,2
      ORL C,3
      MOV FO,C
      .....
      COMPUTE FUNCTION 0
      ANL C, FO
      MOV PLO,C
      .....
      COMPUTE FUNCTION 1
      .....
      ANL C, FO
      MOV P1,1,C
      .....
      COMPUTE FUNCTION 2
      .....
      ANL C, FO
      MOV P1,2,C
      .....

```

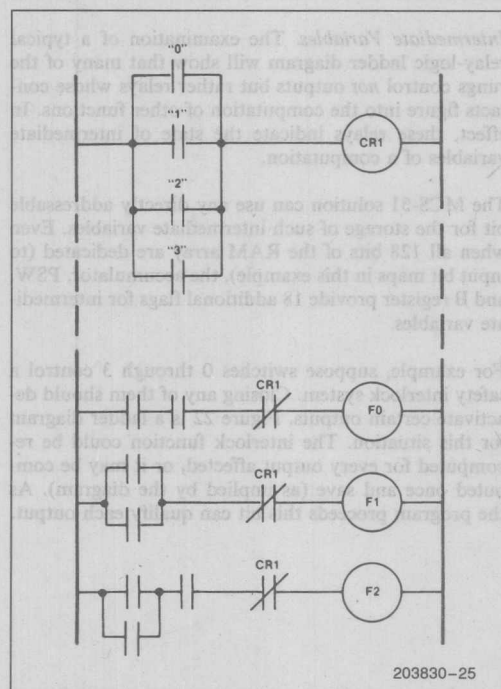


Figure 22. Ladder Diagram for Output Override Circuitry

Latching Relays. A latching relay can be forced into either the ON or OFF state by two corresponding input signals, where it will remain until forced onto the opposite state—analogue to a TTL Set/Reset flip-flop. The relay is used as an intermediate variable for other calculations. In the previous example, the emergency condition could be remembered and remain active until an “emergency cleared” button is pressed.

Any flag or addressable bit may represent a latching relay with a few lines of code (see Example 6).

Example 6. Simulating a latching relay.

```

;I_SET SET FLAG 0 IF C=1
I_SET: ORL C,FO
      MOV FO,C
      ;
;I_RSET RESET FLAG 0 IF C=1
I_RSET: CPL C
      ANL C,FO
      MOV FO,C
      ;

```

Time Delay Relays. A time delay relay does not respond to an input signal until it has been present (or absent) for some predefined time. For example, a ballast or load resistor may be switched in series with a D.C. motor when it is first turned on, and shunted from the circuit after one second. This sort of time delay may be simulated by an interrupt routine driven by one of the two 8051 timer counters. The procedure followed by the routine depends heavily on the details of the exact function needed: time-outs or time delays with resettable or non-resettable inputs are possible. If the interrupt routine is executed every 10 milliseconds the code in Example 7 will clear an intermediate variable set by the background program after it has been active for two seconds.

Example 7. Code to clear USRFLG after a fixed time delay.

```

JNB USR_FLG,NXTTST
DJNZ DLAY_COUNT,NXTTST
CLR USR_FLG
MOV DLAY_COUNT,#200
NXTTST; ;.. .....

```


Serial Interface to Remote Processor. When it detects emergency conditions represented by certain input combinations (such as the earlier Emergency Override), the controller could shut down the machine immediately and/or alert the host processor via the serial port. Code bytes indicating the nature of the problem could be transmitted to a central computer. In fact, at 17,000 bytes-per-second, the entire contents of both bit maps could be sent to the host processor for further analysis in less than a millisecond! If the host decides that conditions warrant, it could alert other remote processors in the system that a problem exists and specify which shut-down sequence each should initiate. For more information on using the serial port, consult the MCS-51 User's Manual.

Response Timing

One difference between relay and programmed industrial controllers (when each is considered as a "black box") is their respective reaction times to input changes. As reflected by a ladder diagram, relay systems contain a large number of "rungs" operating in parallel. A change in input conditions will begin propagating through the system immediately, possibly affecting the output state within milliseconds.

Software, on the other hand, operates sequentially. A change in input states will not be detected until the next time an input scan is performed, and will not affect the outputs until that section of the program is reached. For that reason the raw speed of computing the logical functions is of extreme importance.

Here the Boolean processor pays off. *Every instruction mentioned in this Note* completes in one or two microseconds—the *minimum* instruction execution time for many other microcontrollers! A ladder diagram containing a hundred rungs, with an average of four contacts per rung can be replaced by approximately five hundred lines of software. A complete pass through the entire matrix scanning routine and all computations would require about a millisecond: less than the time it takes for most relays to change state.

A programmed controller which simulates each Boolean function with a subroutine would be less efficient by at least an order of magnitude. Extra software is needed for the simulation routines, and each step takes longer to execute for three reasons: several byte-wide logical instructions are executed per user program step (rather than one Boolean operation); most of those instructions take longer to execute with microprocessors performing multiple off-chip accesses; and calling and returning from the various subroutines requires overhead for stack operations.

In fact, the speed of the Boolean Processor solution is likely to be much faster than the system requires. The CPU might use the time left over to compute feedback parameters, collect and analyze execution statistics, perform system diagnostics, and so forth.

Additional Functions and Uses

With the building-block basics mentioned above many more operations may be synthesized by short instruction sequences.

Exclusive-OR. There are no common mechanical devices or relays analogous to the Exclusive-OR operation, so this instruction was omitted from the Boolean Processor. However, the Exclusive-OR or Exclusive-NOR operation may be performed in two instructions by conditionally complementing the carry or a Boolean variable based on the state of any other testable bit.

```
;EXCLUSIVE-;OR FUNCTION IMPOSED ON CARRY
;USING FO IS INPUT VARIABLE.
;XOR_FO: JNB FO,XORCNT ;("JB" FOR X-NOR)
        CPL C
;XORCNT: ... ..
```

XCH. The contents of the carry and some other bit may be exchanged (switched) by using the accumulator as temporary storage. Bits can be moved into and out of the accumulator simultaneously using the Rotate-

through-carry instructions, though this would alter the accumulator data.

```

;EXCHANGE CARRY WITH USRFLG
XCHBIT: RLC A
        MOV C,USR_FLG
        RRC A
        MOV USR_FLG,C
        RLC A

```

Extended Bit Addressing. The 8051 can directly address 144 general-purpose bits for all instructions in Figure 3b. Similar operations may be extended to any bit anywhere on the chip with some loss of efficiency.

The logical operations AND, OR, and Exclusive-OR are performed on byte variables using six different addressing modes, one of which lets the source be an immediate mask, and the destination any directly addressable byte. Any bit may thus be set, cleared, or complemented with a three-byte, two-cycle instruction if the mask has all bits but one set or cleared.

Byte variables, registers, and indirectly addressed RAM may be moved to a bit addressable register (usually the accumulator) in one instruction. Once transferred, the bits may be tested with a conditional jump, allowing any bit to be polled in 3 microseconds—still much faster than most architectures—or used for logical calculations. (This technique can also simulate additional bit addressing modes with byte operations.)

Parity of bytes or bits. The parity of the current accumulator contents is always available in the PSW, from whence it may be moved to the carry and further processed. Error-correcting Hamming codes and similar applications require computing parity on groups of isolated bits. This can be done by conditionally complementing the carry flag based on those bits or by gathering the bits into the accumulator (as shown in the DES example) and then testing the parallel parity flag.

Multiple byte shift and CRC codes

Though the 8051 serial port can accommodate eight- or nine-bit data transmissions, some protocols involve much longer bit streams. The algorithms presented in

Design Example 2 can be extended quite readily to 16 or more bits by using multi-byte input and output buffers.

Many mass data storage peripherals and serial communications protocols include Cyclic Redundancy (CRC) codes to verify data integrity. The function is generally computed serially by hardware using shift registers and Exclusive-OR gates, but it can be done with software. As each bit is received into the carry, appropriate bits in the multi-byte data buffer are conditionally complemented based on the incoming data bit. When finished, the CRC register contents may be checked for zero by ORing the two bytes in the accumulator.

4.0 SUMMARY

A truly unique facet of the Intel MCS-51 microcomputer family design is the collection of features optimized for the one-bit operations so often desired in real-world, real-time control applications. Included are 17 special instructions, a Boolean accumulator, implicit and direct addressing modes, program and mass data storage, and many I/O options. These are the world's first single-chip microcomputers able to efficiently manipulate, operate on, and transfer either bytes or individual bits as data.

This Application Note has detailed the information needed by a microcomputer system designer to make full use of these capabilities. Five design examples were used to contrast the solutions allowed by the 8051 and those required by previous architectures. Depending on the individual application, the 8051 solution will be easier to design, more reliable to implement, debug, and verify, use less program memory, and run up to an order of magnitude faster than the same function implemented on previous digital computer architectures.

Combining byte- and bit-handling capabilities in a single microcomputer has a strong synergistic effect: the power of the result exceeds the power of byte- and bit-processors laboring individually. Virtually all user applications will benefit in some way from this duality. Data intensive applications will use bit addressing for test pin monitoring or program control flags; control applications will use byte manipulation for parallel I/O expansion or arithmetic calculations.

It is hoped that these design examples give the reader an appreciation of these unique features and suggest ways to exploit them in his or her own application.

ISIS-II MCS-51 MACRO ASSEMBLER V1.0
 OBJECT MODULE PLACED IN: FO: AP70.HEX
 ASSEMBLER INVOKED BY: f1.asm51_ap70_src date(328)

(1000000 INDICATOR ONCE PER SECOND)
 SPECIFICITY: EVI/AGE PROCESSING: 000100

```
LOC OBJ005      LINE      SOURCE
0000      01      $XREF TITLE(AP-70 APPENDIX)
0001      02      ; *****
0002      03      ;
0003      04      ; THE FOLLOWING PROGRAM USES THE BOOLEAN INSTRUCTION SET
0004      05      ; OF THE INTEL 8051 MICROCOMPUTER TO PERFORM A NUMBER OF
0005      06      ; AUTOMOTIVE DASHBOARD CONTROL FUNCTIONS RELATING TO
0006      07      ; TURN SIGNAL CONTROL, EMERGENCY BLINKERS, BRAKE LIGHT
0007      08      ; CONTROL, AND PARKING LIGHT OPERATION.
0008      09      ; THE ALGORITHMS AND HARDWARE ARE DESCRIBED IN DESIGN
0009      10      ; EXAMPLE #4 OF INTEL APPLICATION NOTE AP-70.
0010      11      ; "USING THE INTEL MCS-51(TM)
0011      12      ; BOOLEAN PROCESSING CAPABILITIES"
0012      13      ; *****
0013      14      ;
0014      15      ; INPUT PIN DECLARATIONS:
0015      16      ; (ALL INPUTS ARE POSITIVE-TRUE LOGIC.
0016      17      ; INPUTS ARE HIGH WHEN RESPECTIVE SWITCH CONTACT IS CLOSED.)
0017      18      ;
0018      19      ; BRAKE BIT P1.0 ; BRAKE PEDAL DEPRESSED
0019      20      ; EMERG BIT P1.1 ; EMERGENCY BLINKER ACTIVATED
0020      21      ; PARK BIT P1.2 ; PARKING LIGHTS ON
0021      22      ; L_TURN BIT P1.3 ; TURN LEVER DOWN
0022      23      ; R_TURN BIT P1.4 ; TURN LEVER UP
0023      24      ;
0024      25      ; OUTPUT PIN DECLARATIONS:
0025      26      ; (ALL OUTPUTS ARE POSITIVE TRUE LOGIC.
0026      27      ; BULB IS TURNED ON WHEN OUTPUT PIN IS HIGH.)
0027      28      ;
0028      29      ; L_FRNT BIT P1.5 ; FRONT LEFT-TURN INDICATOR
0029      30      ; R_FRNT BIT P1.6 ; FRONT RIGHT-TURN INDICATOR
0030      31      ; L_DASH BIT P1.7 ; DASHBOARD LEFT-TURN INDICATOR
0031      32      ; R_DASH BIT P2.0 ; DASHBOARD RIGHT-TURN INDICATOR
0032      33      ; L_REAR BIT P2.1 ; REAR LEFT-TURN INDICATOR
0033      34      ; R_REAR BIT P2.2 ; REAR RIGHT-TURN INDICATOR
0034      35      ;
0035      36      ; S_FAIL BIT P2.3 ; ELECTRICAL SYSTEM FAULT INDICATOR
0036      37      ;
0037      38      ; INTERNAL VARIABLE DEFINITIONS:
0038      39      ;
0039      40      ; SUB_DIV DATA 20H ; INTERRUPT RATE SUBDIVIDER
0040      41      ; HI_FREQ BIT SUB_DIV.0 ; HIGH-FREQUENCY OSCILLATOR BIT
0041      42      ; LO_FREQ BIT SUB_DIV.7 ; LOW-FREQUENCY OSCILLATOR BIT
0042      43      ;
0043      44      ; DIM BIT PSW.1 ; PARKING LIGHTS ON FLAG
0044      45      ;
0045      46      ;
0046      47      ;
0047      48      +1 $EJECT
```

203830-26

LOC	OBJ	LINE	SOURCE
		49	ORG 0000H ; RESET VECTOR
0000	020040	50	LJMP INIT ; EVADING FICHL2 ON L1V2
		51	
000B		52	ORG 000BH ; TIMER 0 SERVICE VECTOR
000B	758CF0	53	MOV TH0, #-16 ; HIGH TIMER BYTE ADJUSTED TO CONTROL INT RATE
000E	C0D0	54	PUSH PSW ; EXECUTE CODE TO SAVE ANY REGISTERS USED BELOW
0010	0154	55	AJMP UPDATE ; (CONTINUE WITH REST OF ROUTINE)
		56	
0040		57	ORG 0040H
0040	758A00	58	INIT: MOV TLO, #0 ; ZERO LOADED INTO LOW-ORDER BYTE AND
0043	758CF0	59	MOV TH0, #-16 ; -16 IN HIGH-ORDER BYTE GIVES 4 MSEC PERIOD
0046	758961	60	MOV TMOD, #01100001B ; 8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 0, 16-BIT TIMER MODE FOR TIMER 0 SELECTED
		61	
0049	7520F4	62	MOV SUB_DIV, #244 ; SUBDIVIDE INTERRUPT RATE BY 244 FOR 1 HZ
004C	D2A9	63	SETB ETO ; USE TIMER 0 OVERFLOWS TO INTERRUPT PROGRAM
004E	D2AF	64	SETB EA ; CONFIGURE IE TO GLOBALLY ENABLE INTERRUPTS
0050	D28C	65	SETB TRO ; KEEP INSTRUCTION CYCLE COUNT UNTIL OVERFLOW
0052	80FE	66	SJMP \$; START BACKGROUND PROGRAM EXECUTION
		67	
		68	
0054	D52038	69	UPDATE: DJNZ SUB_DIV, TOSERV ; EXECUTE SYSTEM TEST ONLY ONCE PER SECOND
0057	7520F4	70	MOV SUB_DIV, #244 ; GET VALUE FOR NEXT ONE SECOND DELAY AND
		71	
005A	4390E0	72	ORL P1, #11100000B ; GO THROUGH ELECTRICAL SYSTEM TEST CODE: SET CONTROL OUTPUTS HIGH
005D	43A007	73	ORL P2, #00000111B ; FICHL2 ON
0060	C295	74	CLR L_FRNT ; EMERGENCY FLOAT DRIVE COLLECTOR
0062	20B428	75	JB TO_FAULT ; TO FAULT SHOULD BE PULLED LOW
0065	D295	76	SETB L_FRNT ; PULL COLLECTOR BACK DOWN
0067	C297	77	CLR L_DASH ; REPEAT SEQUENCE FOR L_DASH, ETO
0069	20B421	78	JB TO_FAULT ; TO FAULT
006C	D297	79	SETB L_DASH
006E	C2A1	80	CLR L_REAR ; L_REAR,
0070	20B41A	81	JB TO_FAULT ; TO FAULT
0073	D2A1	82	SETB L_REAR
0075	C296	83	CLR R_FRNT ; SUCCESSIVE CYCLES FILTER R_FRNT,
0077	20B413	84	JB TO_FAULT ; TO FAULT
007A	D296	85	SETB R_FRNT
007C	C2A0	86	CLR R_DASH ; R_DASH AND R_DASH
007E	20B40C	87	JB TO_FAULT ; TO FAULT
0081	D2A0	88	SETB R_DASH ; R_DASH
0083	C2A2	89	CLR R_REAR ; R_REAR AND R_REAR
0085	20B405	90	JB TO_FAULT ; TO FAULT
0088	D2A2	91	SETB R_REAR ; R_REAR
		92	
		93	***** WITH ALL COLLECTORS GROUNDED, TO SHOULD BE HIGH *
		94	***** IF SO, CONTINUE WITH INTERRUPT ROUTINE.
		95	
008A	20B402	96	FAULT: JB TO_TOSERV
008D	82A3	97	CPL S_FAIL ; ELECTRICAL FAILURE PROCESSING ROUTINE
		98	
		99	+1 \$EJECT ; (TOGGLE INDICATOR ONCE PER SECOND)

LOC	OBJ	LINE	SOURCE
		100	CONTINUE WITH INTERRUPT PROCESSING.
		101	
		102	1) COMPUTE LOW BULB INTENSITY WHEN PARKING LIGHTS ARE ON.
		103	
008F	A201	104	TOSERV: MOV C, SUB_DIV 1 ; START WITH 50 PERCENT.
0091	8200	105	ANL C, SUB_DIV 0 ; MASK DOWN TO 25 PERCENT.
0093	7202	106	ORL C, SUB_DIV 2 ; BUILD BACK TO 62.5 PERCENT.
0095	8292	107	ANL C, PARK ; GATE WITH PARKING LIGHT SWITCH.
0097	92D1	108	MOV DIM, C ; AND SAVE IN TEMP. VARIABLE.
		109	
		110	2) COMPUTE AND OUTPUT LEFT-HAND DASHBOARD INDICATOR.
		111	
0099	A293	112	MOV C, L_TURN ; SET CARRY IF TURN
009B	7291	113	ORL C, EMERG ; OR EMERGENCY SELECTED.
009D	8207	114	ANL C, LO_FREQ ; IF SO, GATE IN 1 HZ SIGNAL
009F	9297	115	MOV L_DASH, C ; AND OUTPUT TO DASHBOARD.
		116	
		117	3) COMPUTE AND OUTPUT LEFT-HAND FRONT TURN SIGNAL.
		118	
00A1	92D5	119	MOV FO, C ; SAVE FUNCTION SO FAR.
00A3	72D1	120	ORL C, DIM ; ADD IN PARKING LIGHT FUNCTION
00A5	9295	121	MOV L_FRNT, C ; AND OUTPUT TO TURN SIGNAL.
		122	
		123	4) COMPUTE AND OUTPUT LEFT-HAND REAR TURN SIGNAL.
		124	
00A7	A290	125	MOV C, BRAKE ; GATE BRAKE PEDAL SWITCH
00A9	B073	126	ANL C, /L_TURN ; WITH TURN LEVER.
00AB	72D5	127	ORL C, FO ; INCLUDE TEMP. VARIABLE FROM DASH
00AD	72D1	128	ORL C, DIM ; AND PARKING LIGHT FUNCTION
00AF	92A1	129	MOV L_REAR, C ; AND OUTPUT TO TURN SIGNAL.
		130	
		131	5) REPEAT ALL OF ABOVE FOR RIGHT-HAND COUNTERPARTS.
		132	
00B1	A294	133	MOV C, R_TURN ; SET CARRY IF TURN
00B3	7291	134	ORL C, EMERG ; OR EMERGENCY SELECTED.
00B5	8207	135	ANL C, LO_FREQ ; IF SO, GATE IN 1 HZ SIGNAL
00B7	92A0	136	MOV R_DASH, C ; AND OUTPUT TO DASHBOARD.
00B9	92D5	137	MOV FO, C ; SAVE FUNCTION SO FAR.
00BB	72D1	138	ORL C, DIM ; ADD IN PARKING LIGHT FUNCTION
00BD	9296	139	MOV R_FRNT, C ; AND OUTPUT TO TURN SIGNAL.
00BF	A290	140	MOV C, BRAKE ; GATE BRAKE PEDAL SWITCH
00C1	B094	141	ANL C, /R_TURN ; WITH TURN LEVER.
00C3	72D5	142	ORL C, FO ; INCLUDE TEMP. VARIABLE FROM DASH
00C5	72D1	143	ORL C, DIM ; AND PARKING LIGHT FUNCTION
00C7	92A2	144	MOV R_REAR, C ; AND OUTPUT TO TURN SIGNAL.
		145	
		146	RESTORE STATUS REGISTER AND RETURN.
		147	
00C9	D0D0	148	POP PSW ; RESTORE PSW
00CB	32	149	RETI ; AND RETURN FROM INTERRUPT ROUTINE
		150	
		151	END

203830-29

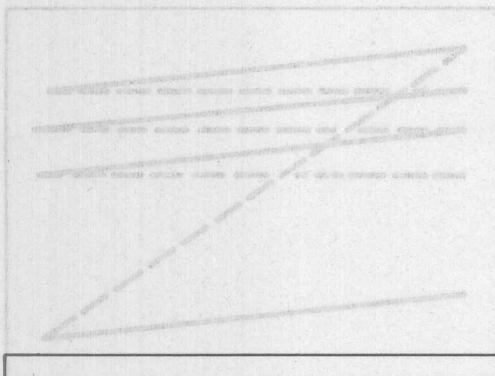


Figure 2.1.0 Raster Scan

As the electron beam moves across the screen under the control of the horizontal oscillator, a third circuit controls the current entering the electron gun. By varying the current, the image may be made as bright or as dim as the user desires. This control is also used to turn the beam off or "blank" the screen.

When the beam reaches the right hand side of the screen, the beam is blanked so it does not appear on the screen as it returns to the left side. This "retrace" of the beam is at a much faster rate than it traveled across the screen and is not visible.

When it takes to scan the whole screen and return to the top left corner, this position is referred to as a "frame". In the case of a commercial television broadcast, a horizontal line is drawn at the top of the screen and a horizontal line is drawn at the bottom. The frame time is equal to the time it takes to scan the whole screen and return to the top left corner.

Although this is the standard, many CRT displays operate from 30KHz horizontal frequency. As the horizontal frequency increases, the number of lines per frame increases. The increase in lines per frame is needed for graphic displays and on special text editors that display many more lines of text than the standard 24 or 25 character lines.

Since the United States operates on a 60Hz A.C. power line frequency, most CRT monitors use 60Hz as the vertical frequency. The use of 60Hz as the vertical frequency allows the magnetic and electronic variations that can modulate the electron beam to be synchronized with the display, thus they go unnoticed. A frequency other than 60Hz is used, special shielding and power supply requirements are needed.

1.0 INTRODUCTION

This is the third application note that Intel has produced on CRT terminal controllers. The first Ap Note (let. 1) written in 1977, used the 8080 as the CPU and required 41 packages including 11 LSI devices. In 1979, another application note (let. 2) using the 8085 as the controller was produced and the chip count decreased to 30 with 11 LSI devices.

Advancing technology has integrated a complete system onto a single device that contains a CPU, program memory, data memory, serial communication, interrupt controller. These "computer-on-a-chip" devices are

October 1984

offer was chosen for this application because of its highly integrated functions. This CRT terminal design uses 12 packages with only 4 LSI devices.

This application note has been divided into five general sections:

- 1) CRT Terminal Basics
- 2) 8051 Description
- 3) 8255 Description
- 4) Design Background
- 5) System Description

2.0 CRT TERMINAL BASICS

A terminal provides a means for humans to communicate with a computer. Terminals may be as simple as a keyboard and a couple of push buttons, or it may be a sophisticated graphics system that contains a function keyboard with user programmable keys, a display, and several processors controlling its functions. This application note describes a basic low cost terminal consisting of a black and white CRT display, full function keyboard, and a serial interface.

2.1 CRT Description

A raster scan CRT displays its images by generating a series of lines (raster) across the face of the tube. The electron beam usually starts at the top left hand corner, moves left to right, back to the left, and then moves down one row and continues right. This is repeated until the lower right hand corner is reached. Then the beam returns to the top left hand corner and retraces the screen. The beam forms a pattern as shown in Figure 2.1.0.

Two independent operating circuits control this movement across the screen. The horizontal oscillator controls the left to right motion of the beam while the vertical oscillator also controls the top to bottom movement. The vertical oscillator also controls the beam when to return to the upper left hand corner.

8051 Based CRT Terminal
Controller
Michael A. Shater
Microcontroller Applications

1.0 INTRODUCTION

This is the third application note that Intel has produced on CRT terminal controllers. The first Ap Note (ref. 1), written in 1977, used the 8080 as the CPU and required 41 packages including 11 LSI devices. In 1979, another application note (ref. 2) using the 8085 as the controller was produced and the chip count decreased to 20 with 11 LSI devices.

Advancing technology has integrated a complete system onto a single device that contains a CPU, program memory, data memory, serial communication, interrupt controller, and I/O. These "computer-on-a-chip" devices are known as microcontrollers. Intel's MCS®-51 microcontroller was chosen for this application because of its highly integrated functions. This CRT terminal design uses 12 packages with only 4 LSI devices.

This application note has been divided into five general sections:

- 1) CRT Terminal Basics
- 2) 8051 Description
- 3) 8276 Description
- 4) Design Background
- 5) System Description

2.0 CRT TERMINAL BASICS

A terminal provides a means for humans to communicate with a computer. Terminals may be as simple as a LED display and a couple of push buttons, or it may be an elaborate graphics system that contains a full function keyboard with user programmable keys, color CRT and several processors controlling its functions. This application note describes a basic low cost terminal containing a black and white CRT display, full function keyboard and a serial interface.

2.1 CRT Description

A raster scan CRT displays its images by generating a series of lines (raster) across the face of the tube. The electron beam usually starts at the top left hand corner moves left to right, back to the left of the screen, moves down one row and continues on to the right. This is repeated until the lower right hand of the screen is reached. Then the beam returns to the top left hand corner and refreshes the screen. The beam forms a zigzag pattern as shown in Figure 2.1.0.

Two independent operating circuits control this movement across the screen. The horizontal oscillator controls the left to right motion of the beam while the vertical controls the top to bottom movement. The vertical oscillator also tells the beam when to return to the upper left hand corner or "home" position.

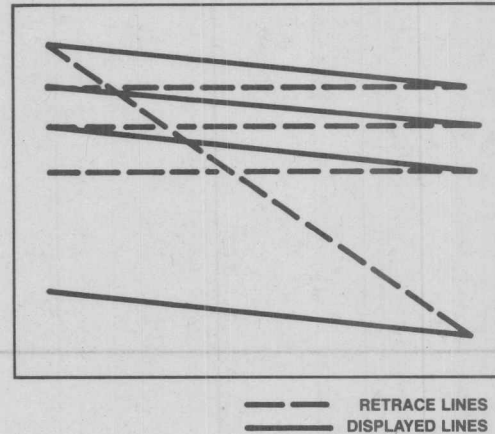


Figure 2.1.0 Raster Scan

As the electron beam moves across the screen under the control of the horizontal oscillator, a third circuit controls the current entering the electron gun. By varying the current, the image may be made as bright or as dim as the user desires. This control is also used to turn the beam off or "blank the screen".

When the beam reaches the right hand side of the screen, the beam is blanked so it does not appear on the screen as it returns to the left side. This "retrace" of the beam is at a much faster rate than it traveled across the screen to generate the image.

The time it takes to scan the whole screen and return to the home position is referred to as a "frame". In the United States, commercial television broadcast uses a horizontal sweep frequency of 15,750Hz which calculates out to 63.5 microseconds per line. The frame time is equal to 16.67 milliseconds or 60Hz vertical sweep frequency.

Although this is the commercial standard, many CRT displays operate from 18KHz to 30KHz horizontal frequency. As the horizontal frequency increases, the number of lines per frame increases. This increase in lines or resolution is needed for graphic displays and on special text editors that display many more lines of text than the standard 24 or 25 character lines.

Since the United States operates on a 60Hz A.C. power line frequency, most CRT monitors use 60Hz as the vertical frequency. The use of 60Hz as the vertical frequency allows the magnetic and electrical variations that can modulate the electron beam to be synchronized with the display, thus they go unnoticed. If a frequency other than 60Hz is used, special shielding and power supply regu-

lating is usually required. Very few CRTs operate on a vertical frequency other than 60Hz due to the increase in the overall system cost.

The CRT controller must generate the pulses that define the horizontal and vertical timings. On most raster scan CRTs the horizontal frequency may vary as much as 500Hz without any noticeable effect on the quality of the display. This variation can change the number of horizontal lines from 256 to 270 per frame.

The CRT controller must also shift out the information to be displayed serially to the circuit that controls the electron beam's intensity as it scans across the screen. The circuits that control the timing associated with the shifting of the information are known as the dot clock and the character clock. The character clock frequency is equal to the dot clock frequency divided by the number of dots it takes to form a character in the horizontal axis. The dot clock frequency is calculated by the following equation:

$$\text{Dot Clock (Hz)} = (N + R) * D * L * F$$

where

- N is the number of displayed characters per row,
- R is the number of character times for the retrace,
- D is the number of dots per character in the horizontal axis,
- L is the number of horizontal lines per frame,
- F is the frame rate in Hz.

In this design N=80, R=20, D=7, L=270, and F=60Hz. Plugging in the numbers results in a dot clock frequency of 11.34MHz.

The retrace number may vary on each design because it is used to set the left and right hand margins on the CRT. The number of dots per character is chosen by the designer to meet the system needs. In this design, a 5x7 dot matrix makes D equal to 5+2=7.

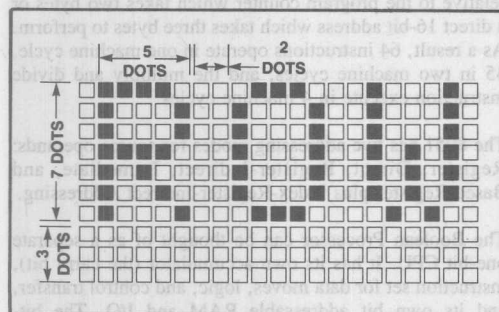


Figure 2.1.1 5 x 7 Dot Matrix

The following equation can be used to figure the number of lines per frame:

$$L = (H * Z) + V$$

where

- H is the number of horizontal lines per character,
- Z is the number of character lines per frame,
- V is the number of horizontal line times during the vertical retrace

In this design H is equal to the 7 horizontal dots per character plus 3 blank dots between each row which adds up to 10. Also 25 lines of characters are displayed, so Z=25. The vertical retrace time is variable to set the top and bottom margins on the CRT and in this design is equal to 20. Plugging in the numbers gives L=270 lines per frame.

2.2 Keyboard

A keyboard is the common way a human enters commands and data to a computer. A keyboard consists of a matrix of switches that are scanned every couple of milliseconds by a keyboard controller to determine if one of the keys has been pressed. Since the keyboard is made up of mechanical switches that tend to bounce or "make and break" contact everytime they are pressed, debouncing of the switches must also be a function of the keyboard controller. There are dedicated keyboard controllers available that do everything from scanning the keyboard, debouncing the keys, decoding the ASCII code for that key closure to flagging the CPU that a valid key has been depressed. The keyboard controller may present the information to the CPU in parallel form or in a serial data stream.

This Application Note integrates the function of the keyboard controller into the 8051 which is also the terminal controller. Provisions have been made to interface the 8051 to a keyboard that uses a dedicated keyboard controller. The 8051 can accept data from the keyboard controller in either parallel or serial format.

2.3 Serial Communications

Communication between a host computer and the CRT terminal can be in either parallel or serial data format. Parallel data transmission is needed in high end graphic terminals where great amounts of information must be transferred.

One can rarely type faster than 120 words per minute, which corresponds to 12 characters per second or 1 character per 83 milliseconds. The utilization of a parallel port cannot justify the cost associated with the drivers and the amount of wire needed to perform this transmission. Full duplex serial data transmission requires 3 wires and two

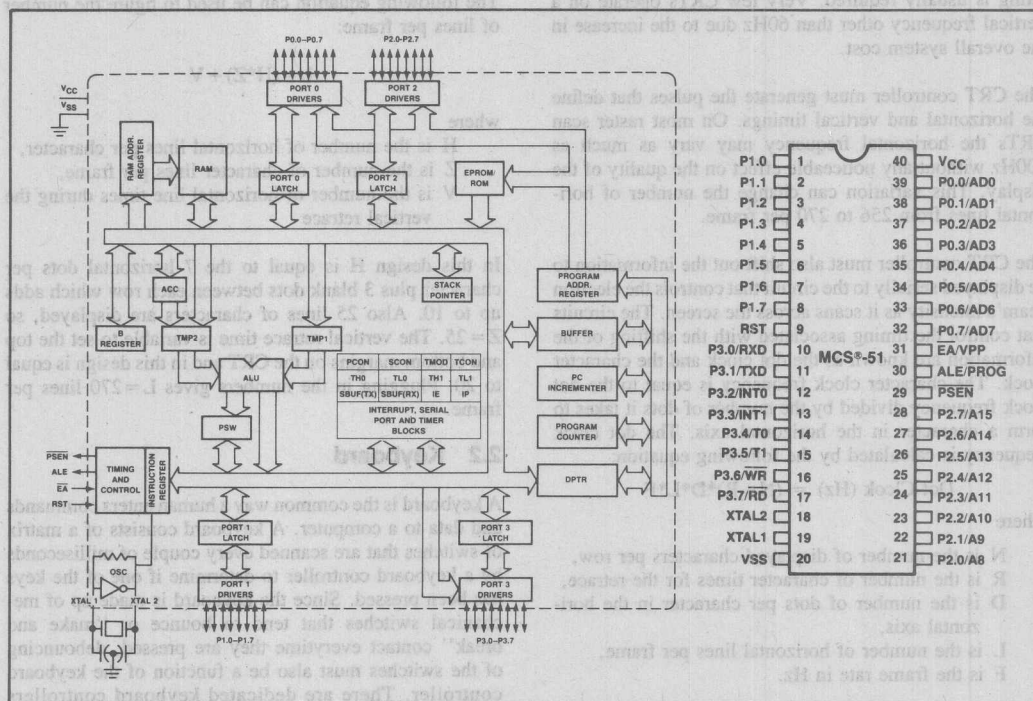


Figure 3.0.0 8051 Block Diagram

drivers to implement the communication channel between the host computer and the terminal. The data rate can be as high as 19200 BAUD in the asynchronous serial format. BAUD rate is the number of bits per second received or transmitted. In the asynchronous serial format, 10 bits of information is required to transmit one character. One character per 500 microseconds or 1,920 characters per second would then be transmitted using 19.2 KBAUD.

This application note uses the 8051 serial port configured for full duplex asynchronous serial data transmission. The software for the 8051 has been written to support variable BAUD rates from 150 BAUD up to 9.6 KBAUD.

3.0 8051 DESCRIPTION

The 8051 is a single chip high-performance microcontroller. A block diagram is shown in figure 3.0.0. The 8051 combines CPU; Boolean processor; 4K \times 8 ROM; 128 \times 8 RAM; 32 I/O lines; two 16-bit timer/ event counters; a five-source, two-priority-level, nested interrupt structure; serial I/O port for either multiprocessor communications, I/O expansion, or full duplex UART; and on-chip oscillator and clock circuits.

3.1 CPU

Efficient use of program memory results from an instruction set consisting of 49 single-byte, 45 two-byte and 17 three-byte instructions. Most arithmetic, logical and branching operations can be performed using an instruction that appends either a short address or a long address. For example, branches may use either an offset that is relative to the program counter which takes two bytes or a direct 16-bit address which takes three bytes to perform. As a result, 64 instructions operate in one machine cycle, 45 in two machine cycles, and the multiply and divide instruction execute in 4 machine cycles.

The 8051 has five addressing modes for source operands: Register, Direct, Register-Indirect, Immediate, and Based-Register-plus Index-Register-Indirect Addressing.

The Boolean Processor can be thought of as a separate one-bit CPU. It has its own accumulator (the carry bit), instruction set for data moves, logic, and control transfer, and its own bit addressable RAM and I/O. The bit-manipulating instructions provide optimum code and speed efficiency for handling on chip peripherals. The

Boolean processor also provides a straight forward means of converting logic equations directly into software. Complex combinational logic functions can be resolved without extensive data movement, byte masking, and test-and-branch trees.

3.2 On-Chip Ram

The CPU manipulates operands in four memory spaces. These are the 64K-byte Program Memory, 64K-byte External Data Memory, 128-byte Internal Data Memory, and 128-byte Special Function Registers (SFRs). Four Register Banks (each with 8 registers), 128 addressable bits, and the Stack reside in the internal Data RAM. The Stack size is limited only by the available Internal Data RAM and its location is determined by the 8-bit Stack Pointer. All registers except for the Program Counter and the four 8-Register Banks reside in the SFR address space. These memory mapped registers include arithmetic registers, pointers, I/O ports, and registers for the interrupt system, timers, and serial channel.

Registers in the four 8-Register Banks can be addressed by Register, Direct, or Register-Indirect Addressing modes. The 128 bytes of internal Data Memory can be addressed by Direct or Register-Indirect modes while the SFRs are only addressed directly.

3.3 I/O Ports

The 8051 has instructions that can treat the 32 I/O lines as 32 individually addressable bits or as 4 parallel 8-bit ports addressable as Ports 0, 1, 2, and 3.

Resetting the 8051 writes a logical 1 to each pin on port 0 which places the output drivers into a high-impedance mode. Writing a logical 0 to a pin forces the pin to ground and sinks current. Re-writing the pin high will place the pin in either an open drain output or high-impedance input mode.

Ports 1, 2, and 3 are known as quasi-bidirectional I/O pins. Resetting the device writes a logical one to each pin. Writing a logical 0 to the pin will force the pin to ground and sink current. Re-writing the pin high will place the pin in an output mode with a weak depletion pullup FET or in the input mode. The weak pullup FET is easily overcome by a TTL output.

Ports 0 and 2 can also be used for off-chip peripheral expansion. Port 0 provides a multiplexed low-order address and data bus while Port 2 contains the high-order address when using external Program Memory or more than 256 byte external Data Memory.

Port 3 pins can also be used to provide external interrupt request inputs, event counter inputs, the serial port TXD

and RXD pins and to generate control signals used for writing and reading external peripherals.

3.4 Interrupt System

External events and the real-time-driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. A five-source, two-level, nested interrupt system ties the real time events to the normal program execution.

The 8051 has two external interrupt sources, one interrupt from each of the two timer/counters, and an interrupt from the serial port. Each interrupt vectors the program execution to its own unique memory location for servicing the interrupt. In addition, each of the five sources can be individually enabled or disabled as well as assigned to one of the two interrupt priority levels available on the 8051.

Up to two additional external interrupts can be created by configuring a timer/counter to the event counter mode. In this mode the timer/counter increments on command by either the T0 or T1 pin. An interrupt is generated when the timer/counter overflows. Thus if the timer/counter is loaded with the maximum count, the next high-to-low transition of the event counter input will cause an interrupt to be generated.

3.5 Serial Port

The 8051's serial port is useful for linking peripheral devices as well as multiple 8051s through standard asynchronous protocols with full duplex operation. The serial port also has a synchronous mode for expansion of I/O lines using shift registers. This hardware serial port saves ROM code and permits a much higher transmission rate than could be achieved through software. The processor merely needs to read or write the serial buffer in response to an interrupt. The receiver is double buffered to eliminate the possibility of overrun if the processor failed to read the buffer before the beginning of the next frame.

The full duplex asynchronous serial port provides the means of communication with standard UART devices such as CRT terminals and printers.

The reader should refer to the microcontroller handbook for a complete discussion of the 8051 and its various modes of operation.

4.0 8276 DESCRIPTION

The 8276's block diagram and pin configuration are shown in Figure 4.0.0. The following sections describe the general capabilities of the 8276.

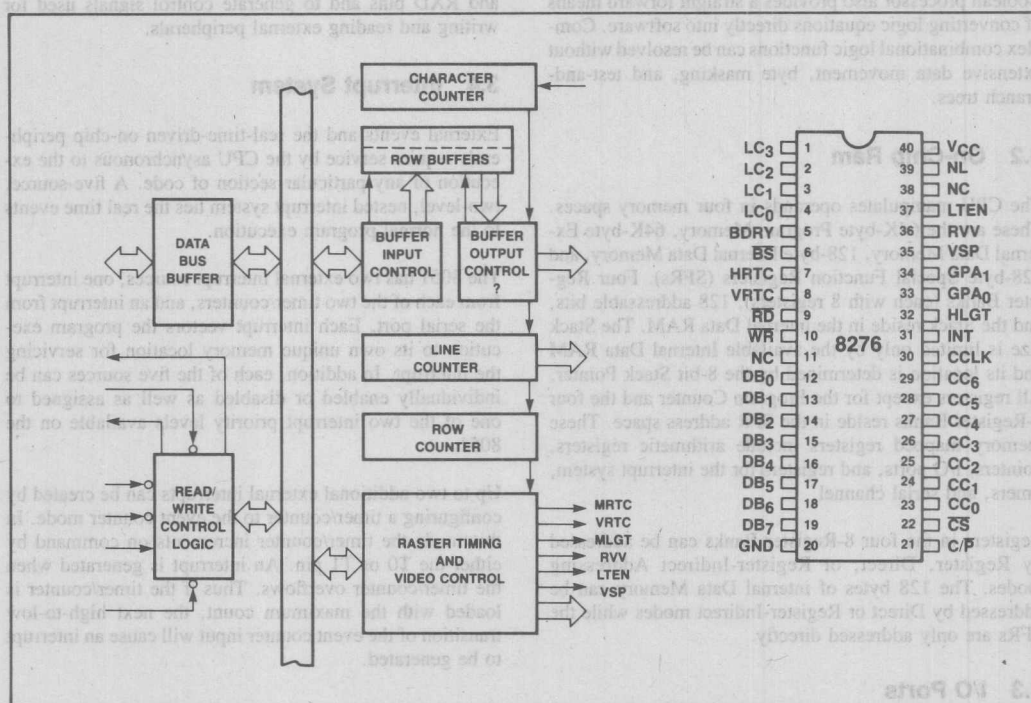


Figure 4.0.0 8276 Block Diagram

4.1 CRT Display Refreshing

The 8276, having been programmed by the system designer for a specific screen format, generates a series of Buffer Ready signals. A row of characters is then transferred by the system controller from the display memory to the 8276's row buffers. The row buffers are filled by deselecting the 8276 CS and asserting the BS and WR signals. The 8276 presents the character codes to an external character generator ROM by using outputs CC0-CC6. The parallel data from the outputs of the character generator is converted to serial information that is clocked by external dot timing logic into the video input of the CRT.

The character rows are displayed on the CRT one line at a time. Line count outputs LC0-LC3 select the current line information from the character generator ROM. The display process is illustrated in Figure 4.1.0. This process is repeated for each display character row. At the beginning of the last display row the 8276 generates an interrupt request by raising its INT output line. The interrupt request

is used by the 8051 system controller to reinitialize its load buffer pointers for the next display refresh cycle.

Proper CRT refreshing requires that certain 8276 parameters be programmed at system initialization time. The 8276 has two types of internal registers; the write only Command (CREG) and Parameter (PREG) Registers, and the read only Status Register (SREG). The 8276 expects to receive a command followed by 0 to 4 parameter bytes depending on the command. A summary of the 8276's instruction set is shown in Figure 4.1.1. To access the registers, CS must be asserted along with WR or RD. The status of the C/P pin determines whether the command or parameter registers are selected.

The 8276 allows the designer flexibility in the display format. The display may be from 1 to 80 characters per row, 1 to 64 rows per screen, and 1 to 16 horizontal lines per character row. In addition, four cursor formats are available; blinking, non-blinking, underline, and reverse video. The cursor position is programmable to anywhere on the screen via the Load Cursor command.

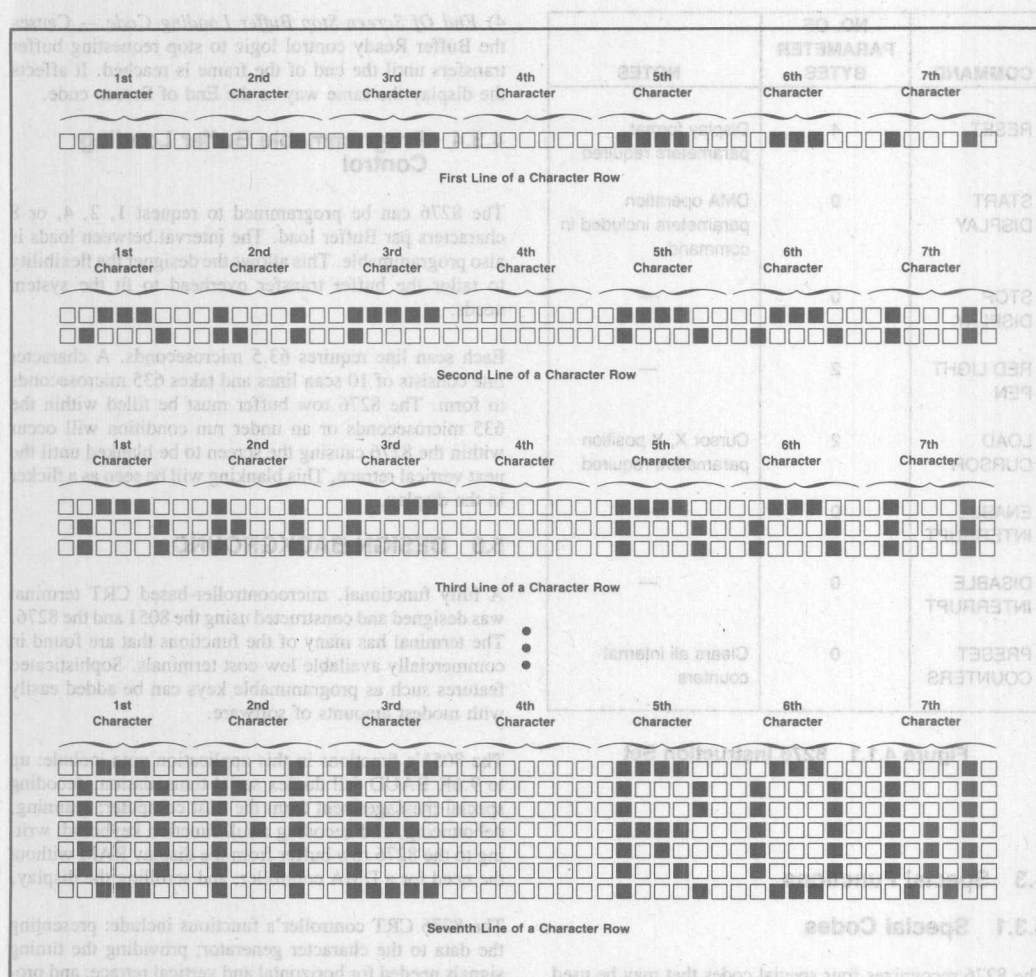


Figure 4.1.0 8276 Row Display

4.2 CRT Timing

The 8276 provides two timing outputs for controlling the CRT. The Horizontal Retrace Timing and Control (HRTC) and Vertical Retrace Timing and Control (VRTC) signals are used for synchronizing the CRT horizontal and vertical oscillators. A third output, VSP (Video Suppress), provides a signal to the dot timing logic to blank the video signal during the horizontal and vertical retraces. LTEN (Light Enable) is used to provide the ability to force the

video output high regardless of the state of the VSP signal. This feature is used to place the cursor on the screen and to control attribute functions.

RVV (Reverse Video) output, if enabled, will cause the system to invert its video output. The fifth timing signal output, HLG (highlight) allows the flexibility to increase the CRT beam intensity to a greater than normal level.

COMMAND	NO. OF PARAMETER BYTES	NOTES
RESET	4	Display format parameters required
START DISPLAY	0	DMA operation parameters included in command
STOP DISPLAY	0	—
RED LIGHT PEN	2	—
LOAD CURSOR	2	Cursor X, Y position parameters required
ENABLE INTERRUPT	0	—
DISABLE INTERRUPT	0	—
PRESET COUNTERS	0	Clears all internal counters

Figure 4.1.1 8276 Instruction Set

4.3 Special Functions

4.3.1 Special Codes

The 8276 recognizes four special codes that may be used to reduce memory, software, or system controller overhead. These characters are placed within the display memory by the system controller. The 8276 performs certain tasks when these codes are received in its row buffer memory.

- 1) *End of Row Code* — Activates VSP. VSP remains active until the end of the line is reached. While VSP is active the screen is blanked.
- 2) *End Of Row-Stop Buffer Loading Code* — Causes the Buffer Ready control logic to stop requesting buffer transfers for the rest of the row. It affects the display the same as End of Row Code.
- 3) *End Of Screen Code* — Activates VSP. VSP remains active until the end of the frame is reached.

4) *End Of Screen-Stop Buffer Loading Code* — Causes the Buffer Ready control logic to stop requesting buffer transfers until the end of the frame is reached. It affects the display the same way as the End of Screen code.

4.3.4 Programmable Buffer Loading Control

The 8276 can be programmed to request 1, 2, 4, or 8 characters per Buffer load. The interval between loads is also programmable. This allows the designer the flexibility to tailor the buffer transfer overhead to fit the system needs.

Each scan line requires 63.5 microseconds. A character line consists of 10 scan lines and takes 635 microseconds to form. The 8276 row buffer must be filled within the 635 microseconds or an under run condition will occur within the 8276 causing the screen to be blanked until the next vertical retrace. This blanking will be seen as a flicker in the display.

5.0 DESIGN BACKGROUND

A fully functional, microcontroller-based CRT terminal was designed and constructed using the 8051 and the 8276. The terminal has many of the functions that are found in commercially available low cost terminals. Sophisticated features such as programmable keys can be added easily with modest amounts of software.

The 8051's functions in this application note include: up to 9.6K BAUD full duplex serial transmission; decoding special messages sent from the host computer; scanning, debouncing, and decoding a full function keyboard; writing to the 8276 row buffer from the display RAM without the need for a DMA controller; and scrolling the display.

The 8276 CRT controller's functions include: presenting the data to the character generator; providing the timing signals needed for horizontal and vertical retrace; and providing blanking and video information.

5.1 Design Philosophy

Since the device count relates to costs, size, and reliability of a system, arriving at a minimum device count without degrading the performance was a driving force for this application note. LSI devices were used where possible to maintain a low chip count and to make the design cycle as short as possible.

PL/M-51 was chosen to generate the majority of the software for this application because it models the human thought process more closely than assembly language. Consequently it is easier and faster to write programs using PL/M-51 and the code is more likely to be correct because less chance exists to introduce errors.

PL/M-51 programs are easier to read and follow than assembly language programs, and thus are easier to modify and customize to the end user's application. PL/M-51 also offers lower development and maintenance costs than assembly language programming.

PL/M-51 does have a few drawbacks. It is not as efficient in code generation relative to assembly language and thus may also run slower.

This application note uses the 8051's interrupts to control the servicing of the various peripherals. The speed of the main program is less critical if interrupts are used. In the last two application notes on terminal controllers, a criterion of the system was the time required for receiving an incoming serial byte, decoding it, performing the function requested, scanning the keyboard, debouncing the keys, and transmitting the decoded ASCII code must be less than the vertical refresh time. Using the 8051 and its interrupts makes this time constraint irrelevant.

5.2 System Target Specifications

The design specifications for the CRT terminal design is as follows:

Display Format

- 80 characters/display row
- 25 display lines

Character Format

- 5 × 7 character contained within a 7 × 10 frame
- First and seventh columns blanked
- Ninth line cursor position
- Programmable delay blinking underline cursor

Control Characters Recognized

- Backspace
- Linefeed
- Carriage Return
- Form Feed

Escape Sequences Recognized

- ESC A, Curser up
- ESC B, Curser down
- ESC C, Curser right
- ESC D, Curser left
- ESC E, Clear screen
- ESC F, Move addressable curser
- ESC H, Home curser
- ESC J, Erase from curser to the end the screen
- ESC K, Erase the current line

Characters Displayed

- 96 ASCII Alphanumeric Characters

Characters Transmitted

- 96 ASCII Alphanumeric Characters
- ASCII Control Character Set
- ASCII Escape Sequence Set
- Auto Repeat

Display Memory

- 2K × 8 static RAM

Data Rate

- Variable rate from 150 to 9600 BAUD

CRT Monitor

- Ball Bros TV-12, 12MHZ Black and White

Keyboard

- Any standard undecoded keyboard (2 key lock-out)
- Any standard decoded keyboard with output enable pin
- Any standard decoded serial keyboard up to 150 BAUD

Scrolling Capability

Compatible With Wordstar

6.0 SYSTEM DESCRIPTION

A block diagram of the CRT terminal is shown in figure 6.0.0. The diagram shows only the essential system features. A detailed schematic of the CRT terminal is contained in the Appendix 7.1.

The "brains" of the CRT terminal is the 8051 microcontroller. The 8276 is the CRT controller in the system, and a 2716 EPROM is used as the character generator. To handle the high speed portion of the CRT, the 8276 is surrounded by a handful of TTL devices. A 2K × 8 static RAM was used as the display memory.

Following the system reset, the 8276 is initialized for curser type, number of characters per line, number of lines, and character size. The display RAM is initialized to all "spaces" (ASCII 20H). The 8051 then writes the "start display" command to the 8276. The local/line input is sampled to determine the terminal mode. If the terminal is on-line, the BAUD rate switches are read and the serial port is set up for full duplex UART mode. The processor then is put into a loop waiting to service the serial port fifo or the 8276.

The serial port is programmed to have the highest priority interrupt. If the serial port generates an interrupt, the processor reads the buffer, puts the character in a generated fifo that resides in the 8051's internal RAM, increments the fifo pointer, sets the serial interrupt flag and returns.

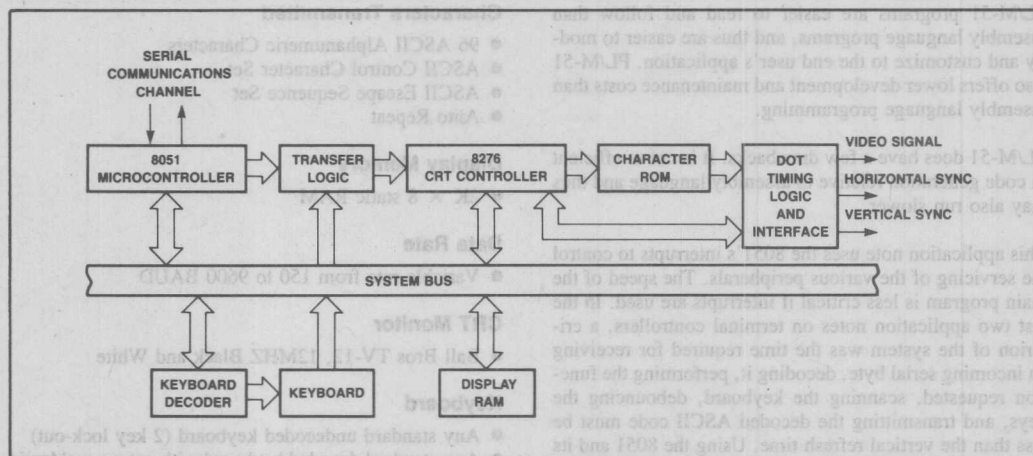


Figure 6.0.0 CRT Terminal Controller Block Diagram

The main program determines if it is a displayable character, a Control word or an ESC sequence and either puts the character in the display buffer or executes the appropriate command sent from the host computer.

If the 8276 needs servicing, the 8051 fills the row buffer for the CRT display's next line. If the 8276 generates a vertical retrace interrupt, the buffer pointers are reloaded with the display memory location that corresponds to the first character of the first display line on the CRT. The vertical retrace also signals the processor to read the keyboard for a key closure.

6.1 Hardware Description

The following section describes the unique characteristics of this design.

6.1.1 Peripheral Address Map

The display RAM, 8276 registers, and the 8276 row buffers are memory mapped into the external data RAM address area. The addresses are as follows:

Read and Write External Display RAM	Address 1000H to 17CFH
Write to 8276 row buffers from Display RAM	Address 1800H to 1FCFH
Write to 8276 Command Register (CREG)	Address 0001H
Write to 8276 Parameter Register (PREG)	Address 0000H
Read from 8276 Status Register (SREG)	Address 0001H

Three general cases can be explored; reading and writing the display RAM, writing to the 8276 row buffers, and reading and writing the 8276's control registers.

As mentioned previously the 8051 fills the 8276 row buffer without the need of a DMA controller. This is accomplished by using a Quad 2-input multiplexor (Figure 6.1.0) as the transfer logic shown in the block diagram. The address line, P2.3, is used to select either of the two inputs. When the address line is low the \overline{RD} and \overline{WR} lines perform their normal functions, that is read and write the

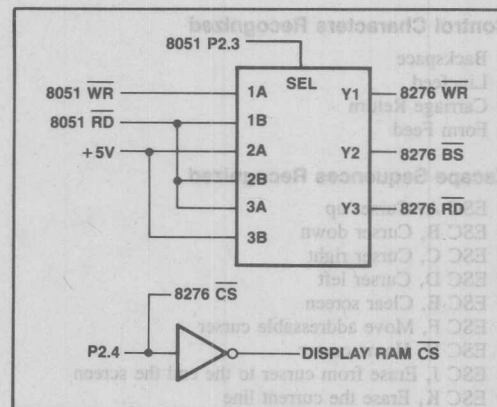


Figure 6.1.0 Simplified Version Of The Transfer Logic

8276 or the external display RAM depending on the states of their respective chip selects. If the address line is high, the 8051 RD line is transformed into \overline{BS} and \overline{WR} signals for the 8276. While holding the address line high, the 8051 executes an external data move (MOVX) from the display RAM to the accumulator which causes the display RAM to output the addressed byte onto the data bus. Since the multiplexor turns the same 8051 RD pulses into \overline{BS} and \overline{WR} pulses to the 8276, the data bus is thus read into the 8276 as a Buffer transfer. This scheme allows 80 characters to be transferred from the display RAM into the 8276 within the required character line time of 635 microseconds. The 8051 easily meets this requirement by accomplishing the task within 350 microseconds.

6.1.2 Scanning The Keyboard

Throughout this project, provision have been made to make the overall system flexible. The software has been written for various keyboards and the user simply needs to link different program modules together to suit their needs.

6.1.2.1 Undecoded Keyboard

Incorporating an undecoded keyboard controller into the other functions of the 8051 shows the flexibility and over all CPU power that is available. The keyboard in this case is a full function, non-buffered 8×8 matrix of switches for a total of 64 possible keys. The 8 send lines are connected to a 3-to-8 open-collector decoder as shown in Figure 6.1.1. Three high order address lines from the 8051 are the decoder inputs. The enabling of the decoder is accomplished through the use of the PSEN signal from the 8051 which makes the architecture of the separate address space for the program memory and the external data RAM work for us to eliminate the need to decode addresses externally. The move code (MOVC) instruction allows each scan line of the keyboard to be read with one instruction.

The keyboard is read by bringing one of the eight scan lines low sequentially while reading the return lines which are pulled high by an external resistor. If a switch is

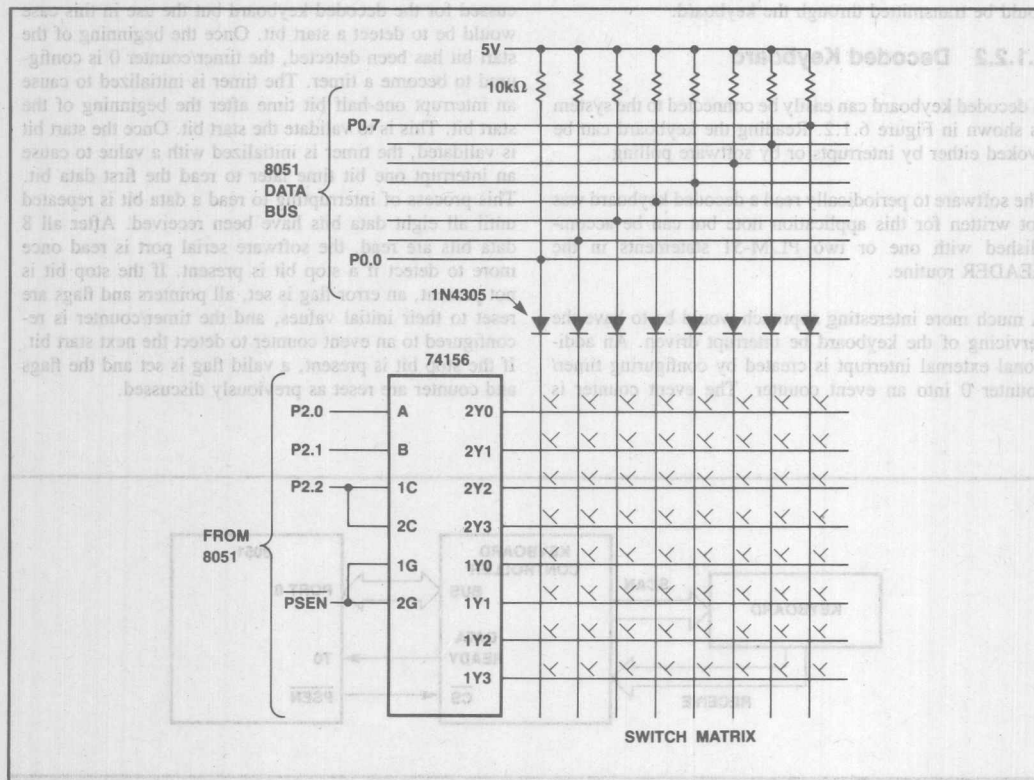


Figure 6.1.1 Keyboard

closed, the data bus line is connected through the switch to the low output of the decoder and one of the data bus lines will be read as a 0. By knowing which scan line detected a key closure and which data bus line was low, the ASCII code for that key can easily be looked up in a matrix of constants. PL/M-51 has the ability to handle arrays and structured arrays, which makes the decoding of the keyboard a trivial task.

Since the Shift, Cap Lock, and Control keys may change the ASCII code for a particular key closure, it is essential to know the status of these pins while decoding the keyboard. The Shift, Cap Lock, and Control keys are therefore not scanned but are connected to the 8051 port pins where they can be tested for closure directly.

The 8 receive lines are connected to the data bus through germanium diodes which chosen for their low forward voltage drop. The diodes keep the keyboard from interfering with the data bus during the times the keyboard is not being read. The circuit consisting of the 3-to-8 decoder and the diodes also offers some protection to the 8051 from possible Electrostatic Discharge (ESD) damage that could be transmitted through the keyboard.

6.1.2.2 Decoded Keyboard

A decoded keyboard can easily be connected to the system as shown in Figure 6.1.2. Reading the keyboard can be evoked either by interrupts or by software polling.

The software to periodically read a decoded keyboard was not written for this application note but can be accomplished with one or two PL/M-51 statements in the `READER` routine.

A much more interesting approach would be to have the servicing of the keyboard be interrupt driven. An additional external interrupt is created by configuring timer/counter 0 into an event counter. The event counter is

initialized with the maximum count. The keyboard controller would inform the 8051 that a valid key has been depressed by pulling the input pin `T0` low. This would overflow the event counter, thus causing an interrupt. The interrupt routine would simply use a `MOVC` (`PSEN` is connected to the output enable pin of the keyboard controller) to read the contents of the keyboard controller onto the data bus, reinitialize the counter to the maximum count and return from the interrupt.

6.1.2.3 Serial Decoded Keyboard

The use of detachable keyboards has become popular among the manufacturers of keyboards and personal computers. This terminal has provisions to use such a keyboard.

The keyboard controller would scan the keyboard, debounce the key and send back the ASCII code for that key closure. The message would be in an asynchronous serial format.

The flowchart for a software serial port is shown in Figure 6.1.3. An additional external interrupt is created as discussed for the decoded keyboard but the use in this case would be to detect a start bit. Once the beginning of the start bit has been detected, the timer/counter 0 is configured to become a timer. The timer is initialized to cause an interrupt one-half bit time after the beginning of the start bit. This is to validate the start bit. Once the start bit is validated, the timer is initialized with a value to cause an interrupt one bit time later to read the first data bit. This process of interrupting to read a data bit is repeated until all eight data bits have been received. After all 8 data bits are read, the software serial port is read once more to detect if a stop bit is present. If the stop bit is not present, an error flag is set, all pointers and flags are reset to their initial values, and the timer/counter is reconfigured to an event counter to detect the next start bit. If the stop bit is present, a valid flag is set and the flags and counter are reset as previously discussed.

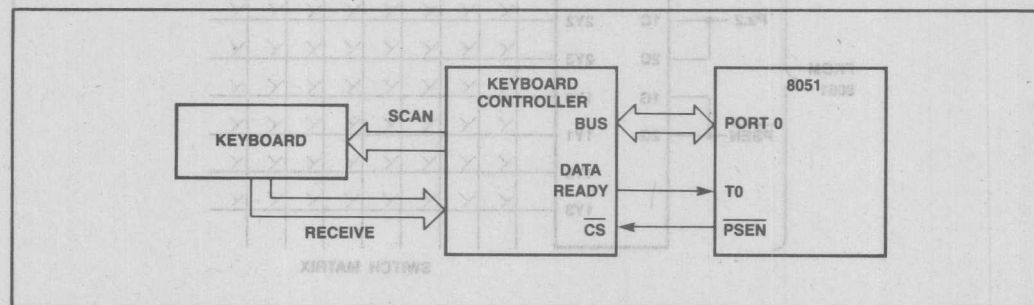


Figure 6.1.2 Using A Decoded Keyboard

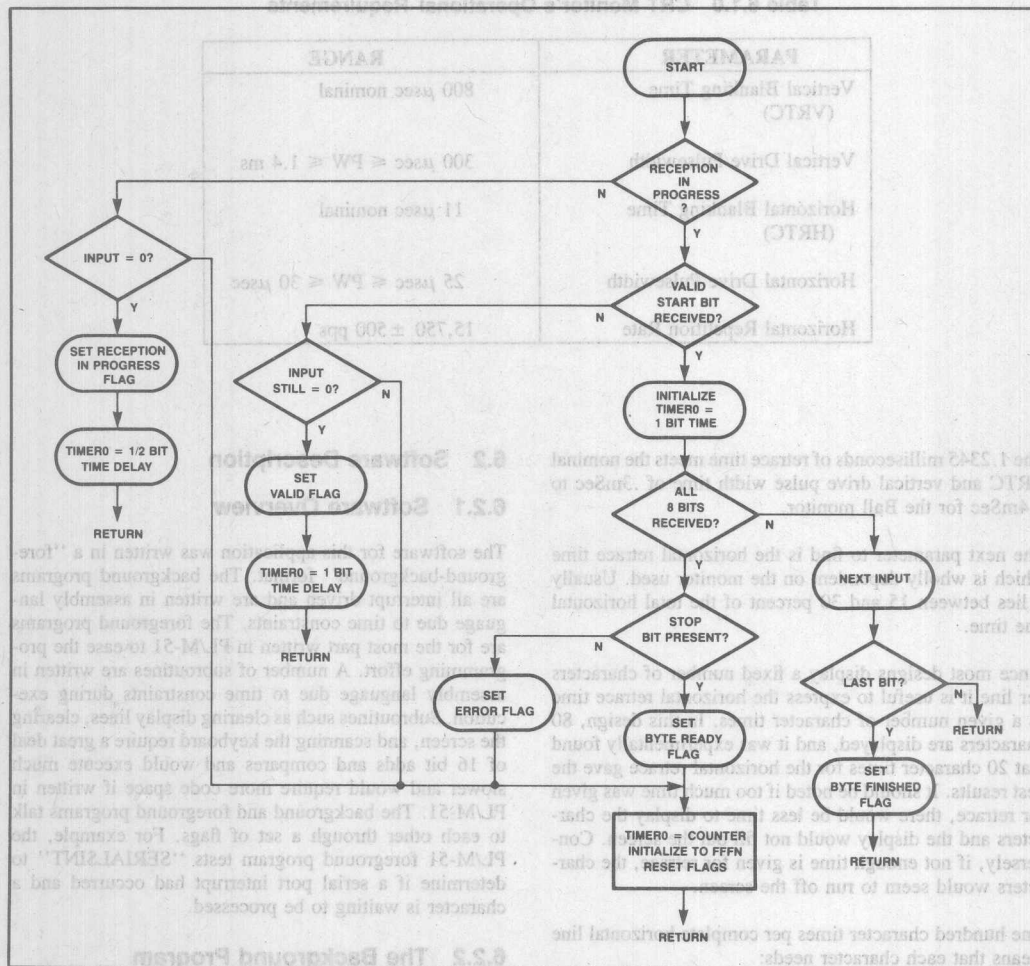


Figure 6.1.3 Flowchart for the Software Serial Port

6.1.4 System Timings

The requirements for the BALL BROTHERS' TV-12 monitor's operation is shown in table 6.1.0. From the monitor's parameters, the 8276 specifications and the system target specifications the system timing is easily calculated.

The 8276 allows the vertical retrace to be only an integer multiple of the horizontal character lines. Twenty-five display lines and a character frame of 7×10 are required from the target specification which will require 250 horizontal lines. If the horizontal frequency is to be within

If we multiply the 20 character times needed to retrace by 61.73 nanoseconds needed for each character, we find 1.2345 milliseconds needed for vertical retrace. To allow for a little more margin at the top and bottom of the screen, two character line times was chosen for the vertical retrace. This choice yields $250 + 20 = 270$ total character lines per frame. Assuming 60 Hz vertical retrace frequency:

$60 \text{ Hz} \times 270 = 16,200 \text{ Hz}$ horizontal frequency
and
 $1/16,200 \text{ Hz} \times 20 \text{ horizontal sync times} = 1.2345 \text{ milliseconds}$

Table 6.1.0 CRT Monitor's Operational Requirements

PARAMETER	RANGE
Vertical Blanking Time (VRTC)	800 μ sec nominal
Vertical Drive Pulsewidth	300 μ sec \leq PW \leq 1.4 ms
Horizontal Blanking Time (HRTC)	11 μ sec nominal
Horizontal Drive Pulsewidth	25 μ sec \leq PW \leq 30 μ sec
Horizontal Repetition Rate	15,750 \pm 500 pps

The 1.2345 milliseconds of retrace time meets the nominal VRTC and vertical drive pulse width time of .3mSec to 1.4mSec for the Ball monitor.

The next parameter to find is the horizontal retrace time which is wholly dependent on the monitor used. Usually it lies between 15 and 30 percent of the total horizontal line time.

Since most designs display a fixed number of characters per line it is useful to express the horizontal retrace time as a given number of character times. In this design, 80 characters are displayed, and it was experimentally found that 20 character times for the horizontal retrace gave the best results. It should be noted if too much time was given for retrace, there would be less time to display the characters and the display would not fill out the screen. Conversely, if not enough time is given for retrace, the characters would seem to run off the screen.

One hundred character times per complete horizontal line means that each character needs:

$$(1/16,200 \text{ Hz}) / 100 \text{ character times} = 617.3 \text{ nanoseconds}$$

If we multiply the 20 character times needed to retrace by 617.3 nanoseconds needed for each character, we find 12.345 microseconds are allocated for retrace. This value falls short of the 25 to 30 microseconds required by the horizontal drive of the Ball monitor. To correct for this, a 74LS123 one-shot was used to extend the horizontal drive pulse width.

The dot clock frequency is easy to calculate now that we know the horizontal frequency. Since each character is formed by seven dots in the horizontal axis, the dot clock period would be the character clock (617.3 nanoseconds) divided by the 7 which is equal to 11.34 MHz. The basic dot timing and CRT timing are shown in the Appendix.

6.2 Software Description

6.2.1 Software Overview

The software for this application was written in a "foreground-background" format. The background programs are all interrupt driven and are written in assembly language due to time constraints. The foreground programs are for the most part written in PL/M-51 to ease the programming effort. A number of subroutines are written in assembly language due to time constraints during execution. Subroutines such as clearing display lines, clearing the screen, and scanning the keyboard require a great deal of 16 bit adds and compares and would execute much slower and would require more code space if written in PL/M-51. The background and foreground programs talk to each other through a set of flags. For example, the PL/M-51 foreground program tests "SERIAL\$INT" to determine if a serial port interrupt had occurred and a character is waiting to be processed.

6.2.2 The Background Program

Two interrupt driven routines, VERT and BUFFER, (see Fig. 6.2.0) request service every 16.67 milliseconds and 617 microseconds respectively. VERT's request comes during the last character row of the display screen. This routine resets the buffer pointers to the first CRT display line in the display memory. VERT is also used as a time base for the foreground program. VERT sets the flag, SCAN, to tell the foreground program (PL/M-51) that it is time to scan the Keyboard. VERT also increments a counter used for the delay between transmitting characters in the AUTO\$REPEAT routine.

The BUFFER routine is executed once per character row. BUFFER uses the multiplexor discussed earlier to fill the 8276's row buffer by executing 80 external data moves and incrementing the Data Pointer between each move.

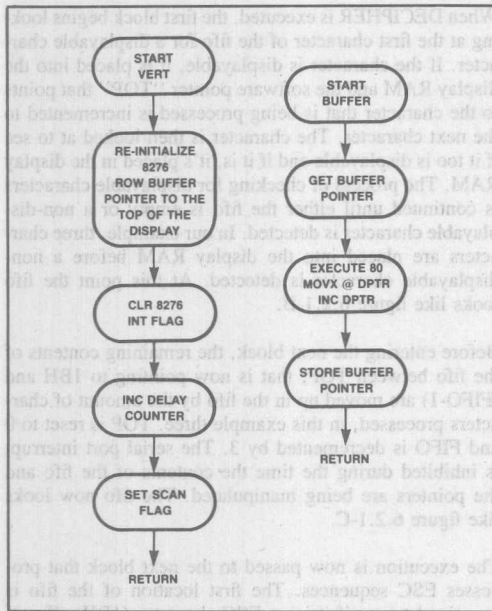


Figure 6.2.0 Flowcharts For VERT and BUFFER Routine

The MOVX reads the display RAM and writes the character into the row buffer during the same instruction.

SERBUF is an interrupt driven routine that is executed each time a character is received or transmitted through the on-chip serial port. The routine first checks if the interrupt was caused by the transmit side of the serial port, signaling that the transmitter is ready to accept another character. If the transmitter caused the interrupt, the flag "TRANSMITSINT" is set which is checked by the foreground program before putting a character in the buffer for transmission.

If the receiver caused the interrupt, the input buffer on the serial port is read and fed into the fifo that has been manufactured in the internal RAM and increments the fifo pointer "FIFO." The flag "SERIALSINT" is then set, telling the foreground program that there is a character in the fifo to be processed. If the read character is an ESC character, the flag "ESCSEQ" is set to tell the foreground program that an escape sequence is in the process of being received.

6.2.3 The Foreground Program

The foreground program is documented in the Appendix. The foreground program starts off by initializing the 8276

as discussed earlier. After all variables and flags are initialized, the processor is put into a loop waiting for either VERT to set SCAN so the program can scan the keyboard, or for the serial port to set SERIALSINT so the program can process the incoming character.

The vertical retrace is used to time the delay between keyboard scans. When VERT gets set, the assembly language routine READER is called. READER scans the keyboard, writing each scan into RAM to be processed later. READER controls two flags, KEY0 and SAME. KEY0 is set when all 8 scans determine that no key is pressed. SAME is set when the same key that was pressed last time the keyboard was read is still pressed.

After READER returns execution to the main program, the flags are tested. If the KEY0 flag is set the main program goes back to the loop waiting for the vertical retrace or a serial port interrupt to occur. If the SAME flag is set the main program knows that the closed key has been debounced and decoded so it sends the already known ASCII code to the AUTO\$REPEAT routine which determines if that character should be transmitted or not.

If KEY0 and SAME are not set, signifying that a key is pressed but it is not the same key as before, the foreground program determines if the results from the scan are valid. First all eight scans are checked to see if only one key was closed. If only one key is closed, the ASCII code is determined, modified if necessary by the Shift, Cap Lock, or Control keys. The NEWSKEY and VALID flags are then set. The next time READER is called, if the same key is still pressed, the SAME flag will be set, causing the AUTO\$REPEAT subroutine to be called as just discussed. Since the keyboard is read during the vertical retrace, 16.67 milliseconds has elapsed between the detection of the pressed key and reverifying that the key is still pressed before transmitting it, thus effectively debouncing the key.

The AUTO\$REPEAT routine is written to transmit any key that the NEWSKEY flag is set for. The counter that is incremented each time the vertical refresh interrupt is serviced causes a programmable delay between the first transmission and subsequent auto repeat transmission. Once the NEWSKEY character is sent, the counter is initialized. Each time the AUTO\$REPEAT routine is called, the counter is checked. Only when the counter overflows will the next character be transmitted. After the initial delay, a character will be transmitted every other time the routine is called as long as the key remains pressed.

6.2.3.1 Handling Incoming Serial Data

One of the criteria for this application note was to make the software less time dependent. By creating a fifo to store incoming characters until the 8051 has time to pro-

cess them, software timing becomes less critical. This application note uses up to 8 levels of the fifo at 9.2KBAUD, and 1 level at 4.8KBAUD and lower. As discussed earlier, the interrupt service routine for the serial port uses the fifo to store incoming data, increments the fifo pointer, "FIFO", and sets SERIAL\$INT to tell the main program that the fifo needs servicing. Once the main program detects that SERIAL\$INT is set the routine DECIPHER is executed.

DECIPHER has three separate blocks; a block for decoding displayable characters, a block for processing Escape sequences, and a block for processing Control codes. Each block works on the fifo independently. Before exiting a block, the contents of the fifo are shifted up by the amount of characters that were processed in that particular block. The shifting of the characters insures that the beginning of the fifo contains the next character to be processed. FIFO is then decremented by the number of characters processed.

Let's look at this process more closely. Figure 6.2.1-A shows a representation of a fifo containing 5 characters. The first three characters in the fifo contain displayable characters, A, B, and C respectively with the last two characters being an ESC sequence for moving the cursor up one line (ESC A) and FIFO points to the next available location to be filled by the serial port interrupt routine; in this case, 5;

When DECIPHER is executed, the first block begins looking at the first character of the fifo for a displayable character. If the character is displayable, it is placed into the display RAM and the software pointer "TOP" that points to the character that is being processed is incremented to the next character. The character is then looked at to see if it too is displayable and if it is, it's placed in the display RAM. The process of checking for displayable characters is continued until either the fifo is empty or a non-displayable character is detected. In our example, three characters are placed into the display RAM before a non-displayable character is detected. At this point the fifo looks like figure 6.2.1-B.

Before entering the next block, the remaining contents of the fifo between TOP, that is now pointing to 1BH and (FIFO-1) are moved up in the fifo by the amount of characters processed, in this example three. TOP is reset to 0 and FIFO is decremented by 3. The serial port interrupt is inhibited during the time the contents of the fifo and the pointers are being manipulated. The fifo now looks like figure 6.2.1-C.

The execution is now passed to the next block that processes ESC sequences. The first location of the fifo is examined to see if it is an ESC character (1BH). If not, the execution is passed to the next block of DECIPHER that processes Control codes. In this case the fifo does contain an ESC code. The flag ESC\$SEQ is checked to see if the 8051 is in the process of receiving an ESC sequence thus signifying that the next byte of the sequence has not been received yet. If the ESC\$SEQ is not set, the next character in the fifo is checked for a valid escape code and the proper subroutine is then called. The fifo contents are then shifted as discussed for the previous block. Due to the length of time that is needed to execute an ESC code sequence or a Control code, only one ESC code and/or Control code can be processed each time DECIPHER is executed.

If at the end of the DECIPHER routine, FIFO contains a 0, the flag SER\$INT is reset. If SER\$INT remains set, DECIPHER will be executed immediately after returning to the main program if SCAN had not been set during the execution of the DECIPHER routine, otherwise DECIPHER will be called after the keyboard is read.

6.2.4 Memory Pointers and Scrolling

The cursor always points to the next location in display memory to be filled. Each time a character is placed in the display memory, the cursor position needs to be tested to determine if the cursor should be incremented to the beginning of the next line of the display or simply moved to the next position on the current display line. The cursor position pointers are then updated in both the 8276 and the internal registers in the 8051.

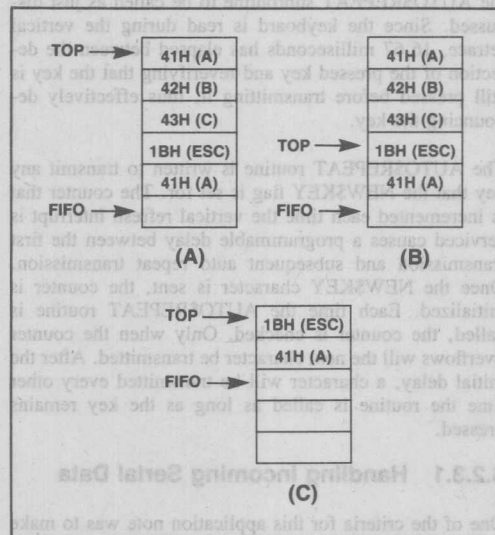


FIGURE 6.2.1 FIFO

When the 2000th character is entered into the display memory, a full display page has been reached signaling the need for the display to scroll. The memory pointer that points to the display memory that contains the first character of the first display line, LINE0, prior to scrolling contains 1800H which is the starting address of the display memory. Each scrolling operation adds 80 (50H) to LINE0 which will now point to the following row in memory as shown in figure 6.2.2-B. LINE0 is used during the vertical

refresh routine to re-initialize the pointers associated with filling the 8276 row buffers.

The display memory locations that were the first line of the CRT display now becomes the last line of the CRT display. Incoming characters are now entered into the display memory starting with 1800H, which is now the first character of the last line of the display screen.

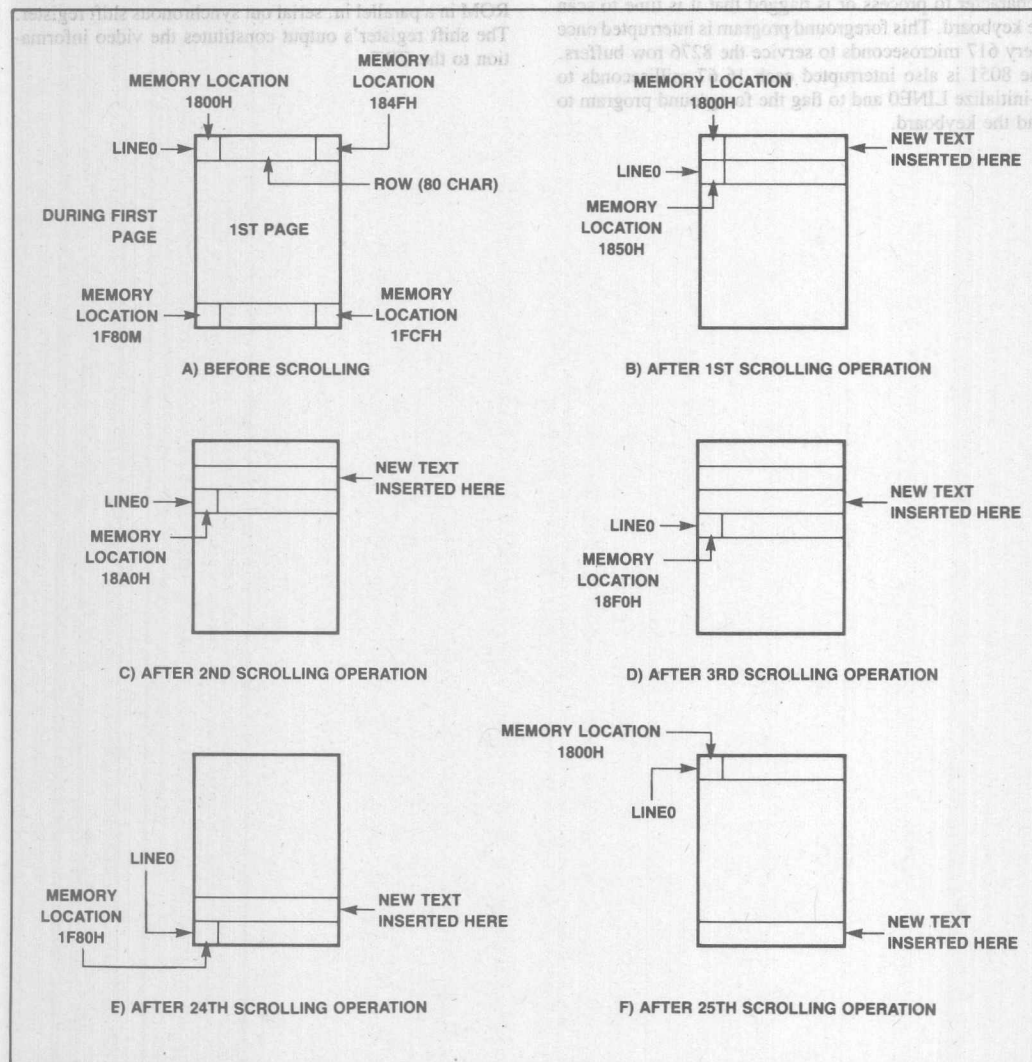


Figure 6.2.2 Pointer Manipulation During Scrolling

6.2.5 Software Timing

The use of interrupts to tie the operation of the foreground program to the real-time events of the background program has made the software timing non-critical for this system.

6.3 System Operation

Following the system reset, the 8051 initializes all on-chip peripherals along with the 8276 and display ram. After initialization, the processor waits until the fifo has a character to process or is flagged that it is time to scan the keyboard. This foreground program is interrupted once every 617 microseconds to service the 8276 row buffers. The 8051 is also interrupted each 16.67 milliseconds to re-initialize LINE0 and to flag the foreground program to read the keyboard.

As discussed earlier, a special technique of rapidly moving the contents of the display RAM to the 8276 row buffers without the need of a DMA device was employed. The characters are then synchronously transferred to the character generator via CC0-CC6 and LC0-LC2 which are used to display one line at a time. Following the transfer of the first line to the dot timing logic, the line count is incremented and the second line is selected. This process continues until the last line of the character is transferred.

The dot timing logic latches the output of the character ROM in a parallel in, serial out synchronous shift register. The shift register's output constitutes the video information to the CRT.

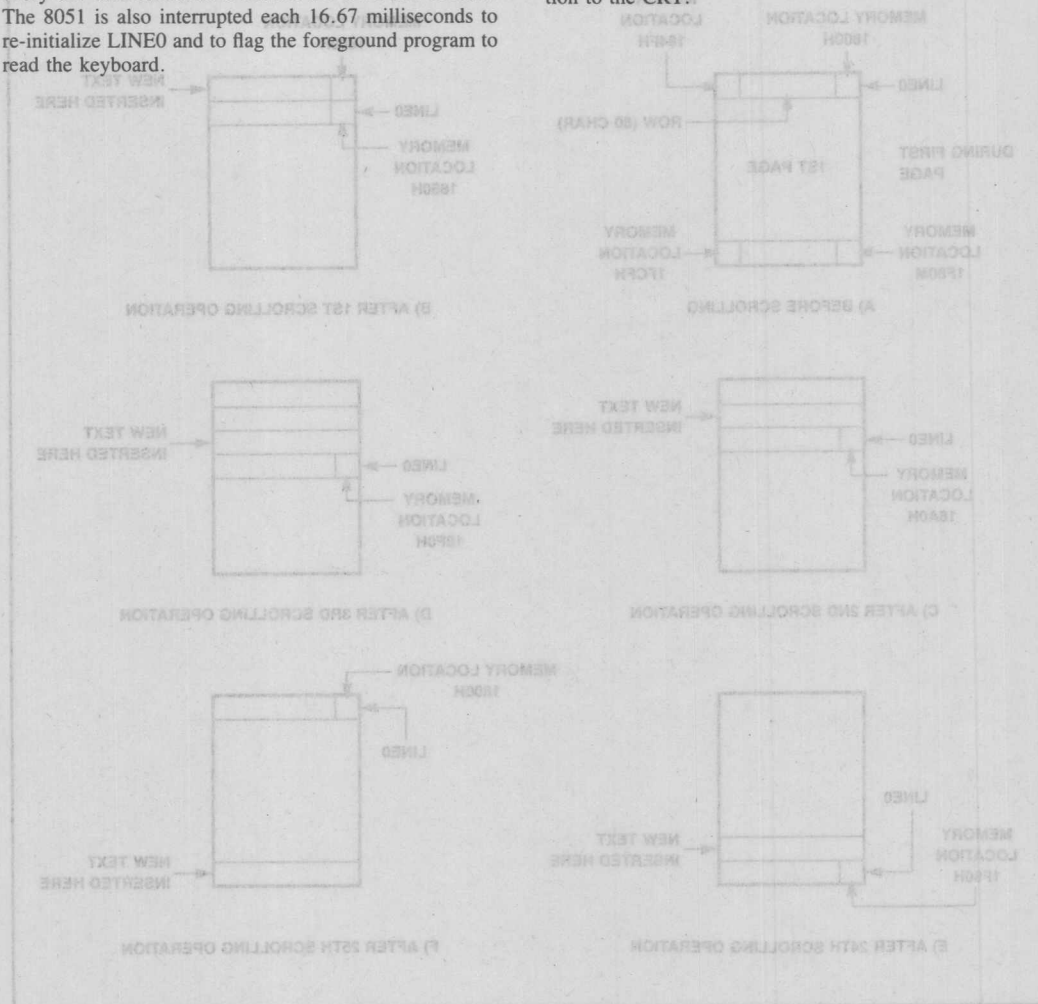
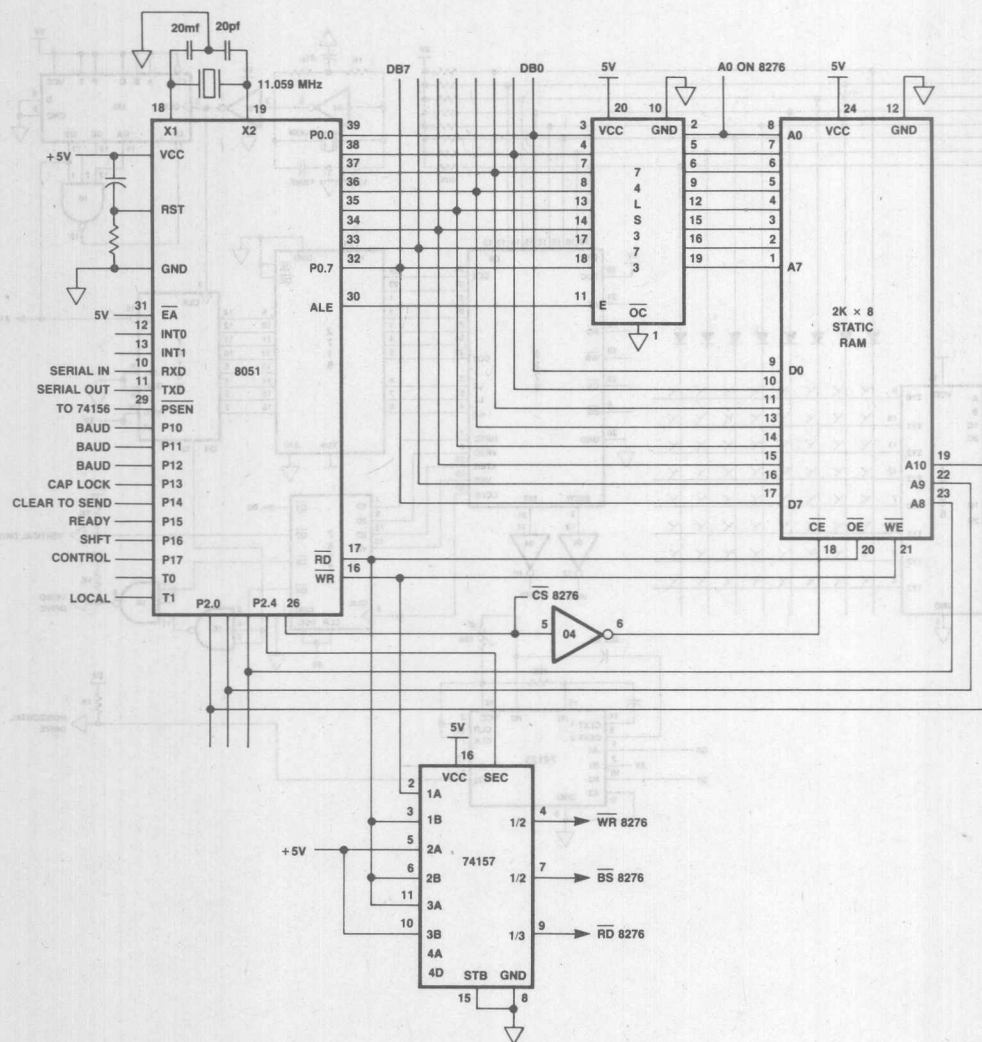
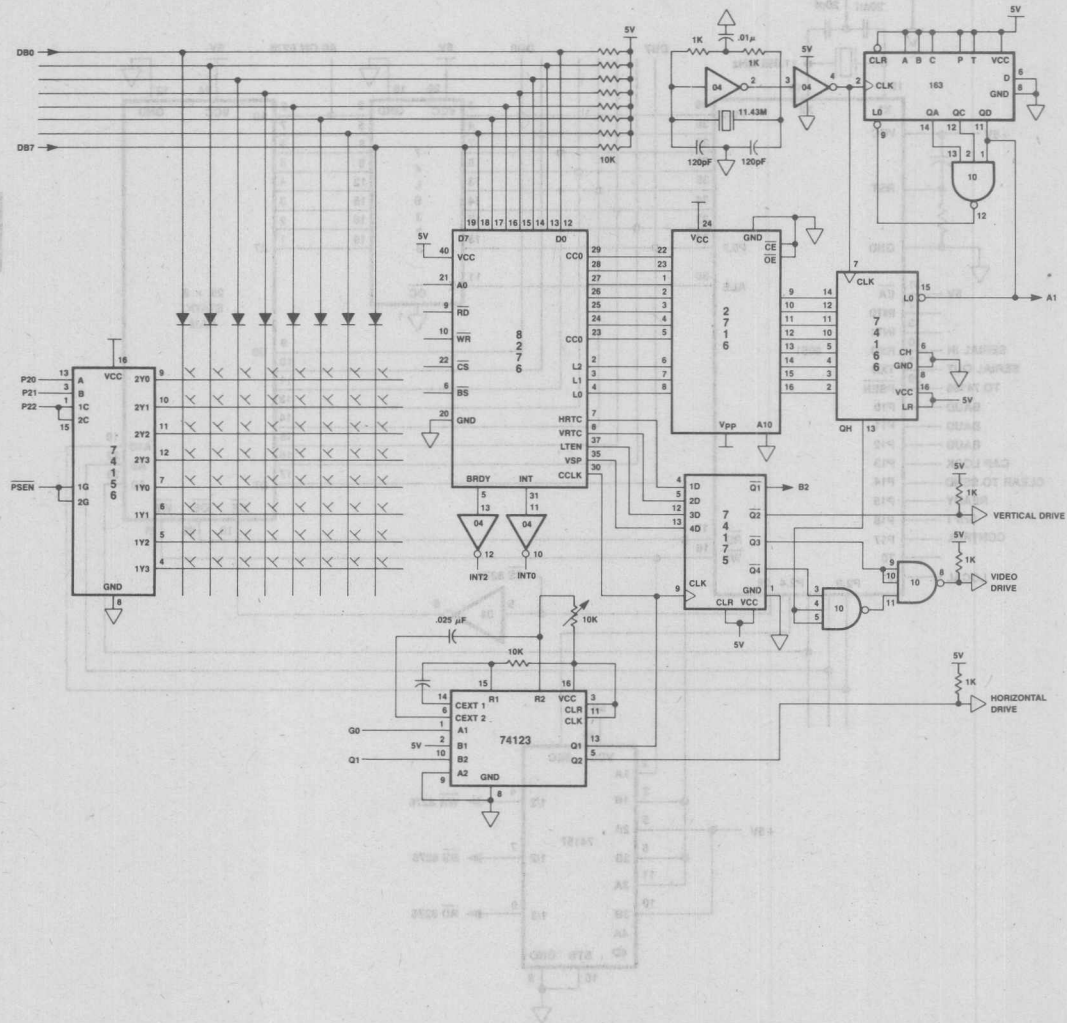


Figure 8.2.2 Pointer Manipulation During Scrolling

Appendix 7.1 CRT Schematics

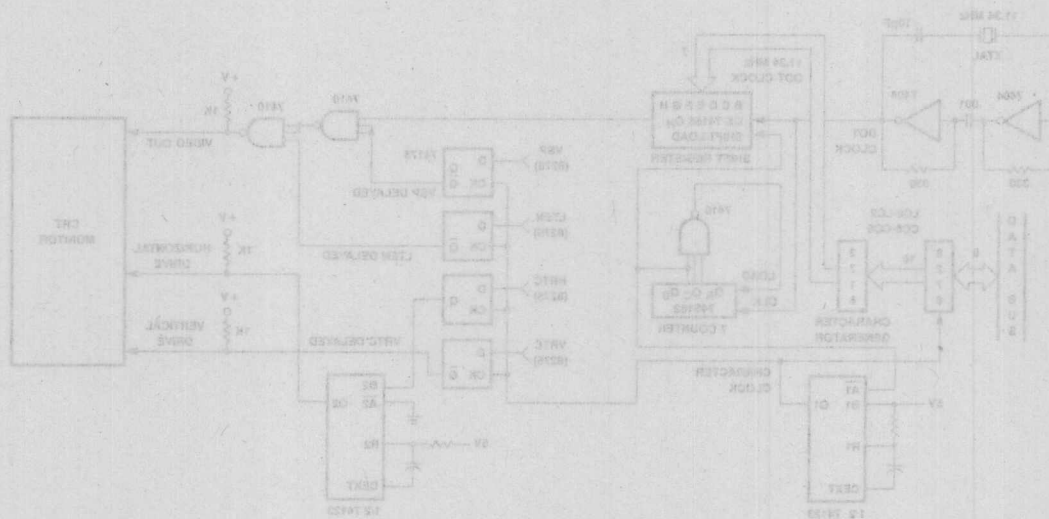
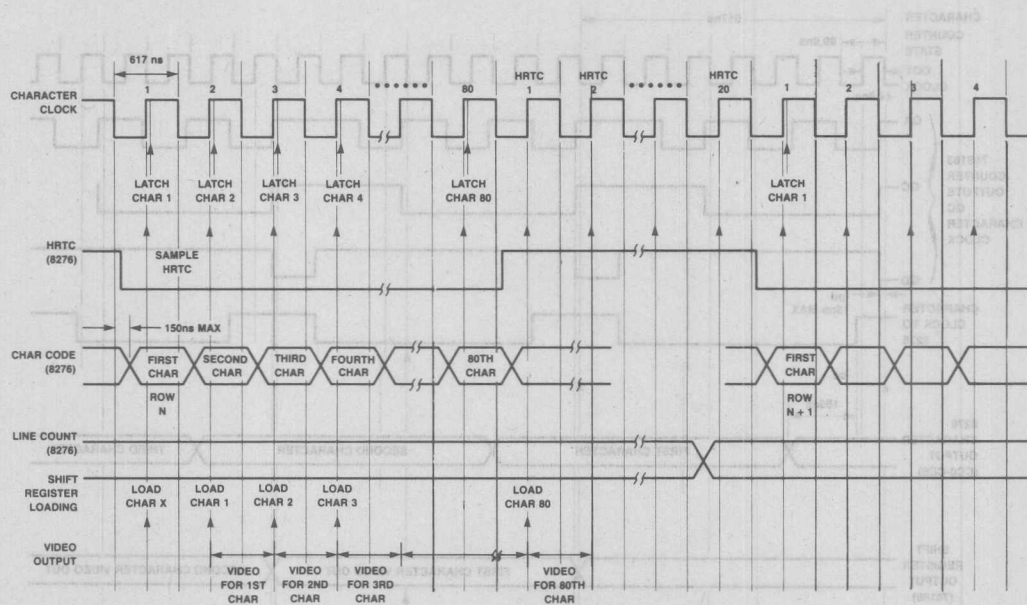






Appendix 7.3 CRT System Timing

Appendix 7.3 Dot Timing



Appendix 7.4 Escape/Control/Display Character Summary

BIT	CONTROL CHARACTERS				DISPLAYABLE CHARACTER				ESCAPE SEQUENCE					
	000	001	010	011	100	101	110	111	010	011	100	101	110	111
0000	NUL @	DLE P	SP		@	P		P						
0001	SOH A	DC1 Q	1	1	A	Q	A	Q			↑ A			
0010	STX B	DC2 R		2	B	R	B	R			↓ B			
0011	ETX C	DC3 S	=	3	C	S	C	S			→ C			
0100	EOT D	DC4 T	\$	4	D	T	D	T			← D			
0101	ENQ E	NAK U	%	5	E	U	E	U			CLR E			
0110	ACK F	SYN V	&	6	F	V	F	V						
0111	BEL G	ETB W		7	G	W	G	W						
1000	BS H	CAN X	(8	H	X	H	X			HOME H			
1001	HT I	EM Y)	9	I	Y	I	Y						
1010	LF J	SUB Z	*	:	J	Z	J	Z			EOS I			
1011	VT K	ESC	+	;	K	[K				EL J			
1100	FF L	FS	,		L		L							
1101	CR M	GS	-	=	M]	M							
1110	SO N	RS	.		N	^	N							
1111	S1 O	US -	/	?	O	-	O							

NOTE: Shaded blocks — functions terminal will react to. Others can be generated but are ignored upon receipt.

Appendix 7.5 Character Generator

As previously mentioned, the character generator used in this terminal is a 2716 EPROM. A 1K by 8 device would have been sufficient since a 128 character 5 by 7 dot matrix only requires 8K of memory. A custom character set could have been stored in the second 1K bytes of the 2716. Any of the free I/O pins on the 8051 could have been used to switch between the character sets.

The three low-order line count outputs (LC0-LC2) from the 8276 are connected to the three low-order address lines of the character generator. The CC0-CC6 output lines are connected to the A3-A9 lines of the character generator.

The output of the character generator is loaded into the shift register. The serial output of the shift register is the video output to the CRT.

Let's assume that the letter "E" is to be displayed. The ASCII code for "E" (45H) is presented to the address lines A2-A9 of the character generator. The scan lines (LC0-LC2) will now count from 0 to seven to form the character as shown in Figure 7.5.0. The same procedure is used to form all 128 possible characters. For reference Appendix 7.6 contains the HEX dump of the character generator used in this terminal.

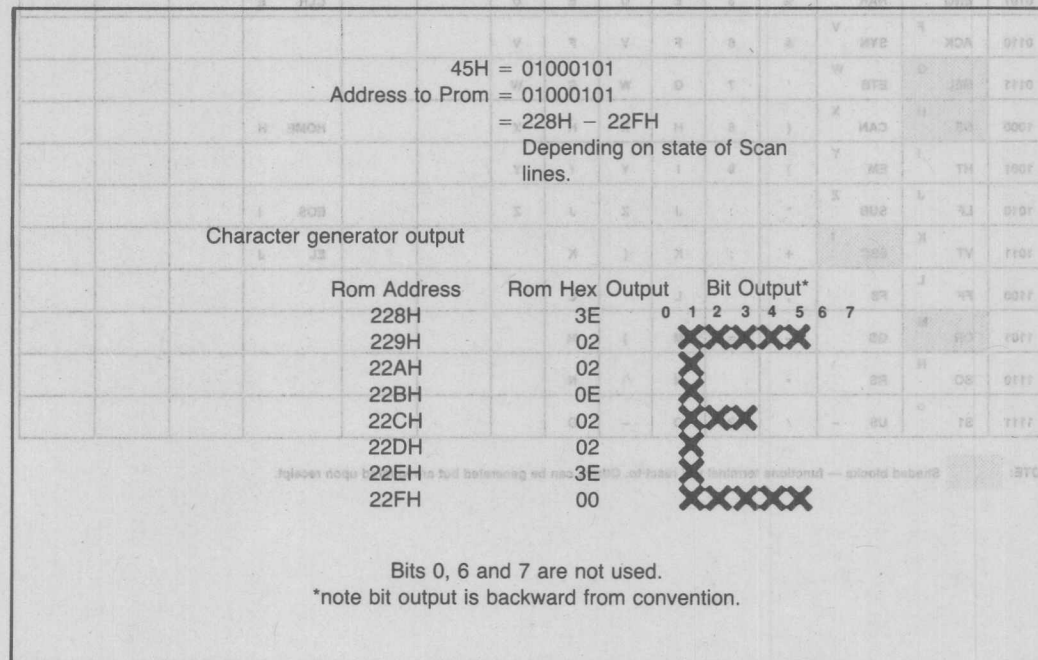


Figure 7.5.0 Character Generator

2

Appendix 7.7 Composite Video

In this design it was assumed that the CRT monitor required a separate horizontal drive, vertical drive, and video input. Many monitors require a composite video signal. The schematic shown in Figure 7.7.0 illustrate how to generate a composite video from the output of the 8276.

The dual one-shots are used to provide a small delay and the proper horizontal and vertical pulse to the composite video monitor. The delay introduced in the horizontal and vertical timing is used to center the display. The 7486 is used to mix the vertical and horizontal retrace. Q1 mix the video and retrace signals along with providing the proper D.C. levels.

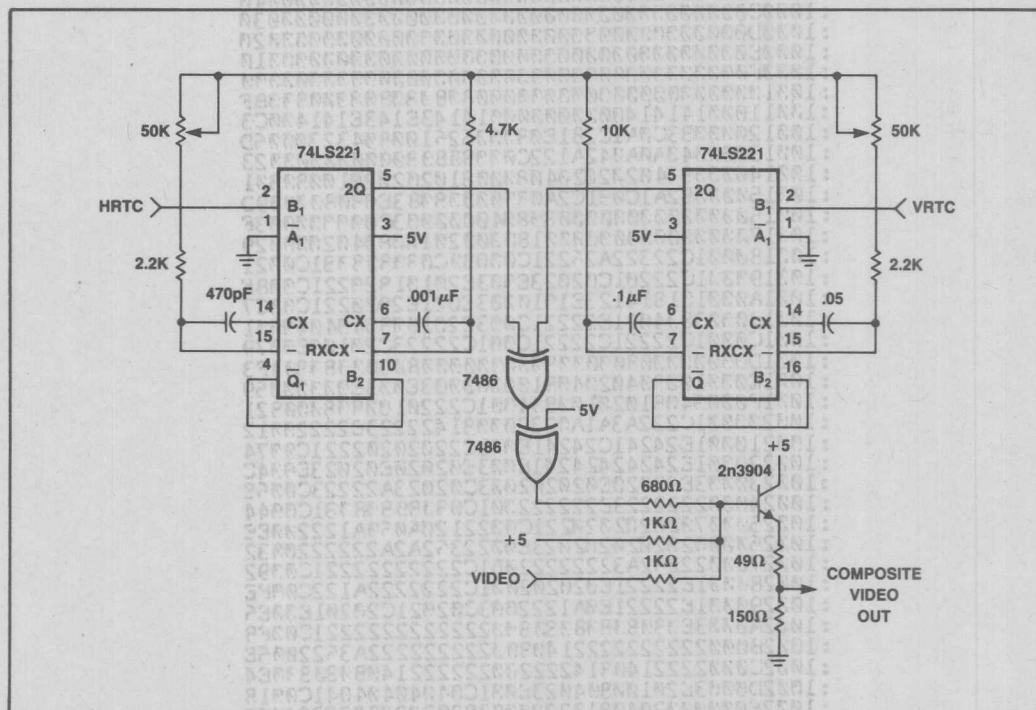


Figure 7.7.0 Composite Video

Appendix 7.8 Software Documentation

```
*****  
*****  
*****  
***** SOFTWARE DOCUMENTATION FOR THE 8051 *****  
***** TERMINAL CONTROLLER APPLICATION NOTE *****  
*****  
*****  
*****
```

MEMORY MAP ASSOCIATED WITH PERIPHERAL DEVICES (USING MOVX):

```

8051 WR AND READ DISPLAY RAM- ADDRESS 1000H TO 17CFH
8051 WR DISPLAY RAM TO THE 8276- ADDRESS 1800H TO 1FCFH
8276 COMMAND ADDRESS- ADDRESS 0001H
8276 PARAMETER ADDRESS- ADDRESS 0000H
8276 STATUS REGISTER- ADDRESS 0001H

```

MEMORY MAP FOR READING THE KEYBOARD (USING MOVC):

KEYBOARD ADDRESS- ADDRESS 10FFH TO 17FFH

```

/***** START MAIN PROGRAM *****/

```

```
/* BEGIN BY PUTTING THE ASCII CODE FOR BLANK IN THE DISPLAY RAM*/
```

```
INIT:
{FILL 2000 LOCATIONS IN THE DISPLAY RAM WITH SPACES (ASCII 20H)}
```

```
/* INITIALIZE POINTERS, RAM BITS, ETC.
```

```
{INITIALIZE POINTERS AND FLAGS}
{INITIALIZE TOP OF THE CRT DISPLAY "LINE0"=1800H}
{INITIALIZE 8276 BUFFER POINTER "RASTER" =1800H}
{INITIALIZE DISPLAYSRAMSPONTER=0000H}
```

```
/* INITIALIZE THE 8276 */
```

```

[ RESET THE 8276]
[ INITIALIZE 8276 TO 80 CHARACTER/ROW }
[ INITIALIZE 8276 TO 25 ROWS PER FRAME }
[ INITIALIZE 8276 TO 10 LINES PER ROW }
[ INITIALIZE 8276 TO NON-BLINKING UNDERLINE CURSER }
[ INITIALIZE CURSER TO HOME POSITION (00,00) (UPPER LEFT HAND CORNER) }
[ START DISPLAY }
[ ENABLE 8276 INTERRUPT ]

```

/* SET UP 8051 INTERRUPTS AND PRIORITIES */

```
{SERIAL PORT HAS HIGHEST INTERRUPT PRIORITY}
{EXTERNAL INTERRUPTS ARE EDGE SENSITIVE}
{ENABLE EXTERNAL INTERRUPTS}
```

/*PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT WILL BE TRANSMITTED TO THE HOST COMPUTER.*/

SCANNER:

```

{ENABLE 8051 GLOBAL INTERRUPT BIT}

/* PROGRAMMABLE DELAY FOR THE CURSER BLINK */

IF {30 VERTICAL RETRACE INTERRUPTS HAVE OCCURRED (CURSER$COUNT=1FH)} THEN
DO;
  {COMPLEMENT CURSER$ON}
  {CLEAR CURSER$COUNT}
  IF {CURSER IS TO BE OFF (CURSER$ON=0)} THEN {MOVE CURSER OFF THE SCREEN}
  CALL LOAD$CURSER;
END;

IF {THE LOCAL$LINE SWITCH HAS CHANGED STATE} THEN
DO;
  IF {IN LOCAL MODE} THEN {DISABLE SERIAL PORT INTERRUPT}
  ELSE CALL CHECK$BAUD$RATE;
END;

DO WHILE {INBETWEEN VERTICAL REFRESHES}
  IF {THE FIFO HAS A CHARACTER TO PROCESS (SERIAL$INT=1)} THEN CALL DECIPHER;
END;

CALL READER;

IF {THE PRESENT PRESSED KEY IS EQUAL TO THE LAST KEY PRESSED AND VALID=1} THEN
  CALL AUTO$REPEAT;
ELSE
DO;
  IF {A KEY IS PRESSED BUT NOT THE SAME ONE AS THE LAST KEYBOARD SCAN} THEN
  DO;
    IF {ONLY ONE KEY IS PRESSED} THEN
      {GET THE ASCII CODE FOR IT}
      {SET NEWSKEY AND VALID FLAGS}
    ELSE {RESET VALID AND NEWSKEY FLAGS}
  END;
  ELSE {THE KEYBOARD MUST NOT HAVE A KEY PRESSED SO RESET VALID$KEY AND NEWSKEY FLAGS}
END;

GOTO SCANNER;
END;

/* PROCEDURE AUTO$REPEAT: THIS PROCEDURE WILL PERFORM AN AUTO-REPEAT FUNCTION
BY TRANSMITTING A CHARACTER EVERY OTHER TIME THIS ROUTINE IS CALLED.
THE AUTO REPEAT FUNCTION IS ACTIVATED AFTER A FIXED DELAY PERIOD AFTER THE
FIRST CHARACTER IS SENT*/

```

AUTO\$REPEAT:

```

IF {THE KEY PRESSED IS NEW (NEWSKEY=1)} THEN
DO;
  {CLEAR THE DIVIDE BY TWO COUNTER "TRANSMIT$TOGGLE"}
  {INITIALIZE THE DELAY COUNTER "TRANSMIT$COUNT" TO 0D0H}
  CALL TRANSMIT;
  {CLEAR NEWSKEY}
END;

```

```

ELSE
DO;
  IF {TRANSMIT$COUNT IS NOT EQUAL TO 0} THEN
  DO;
    {INCREMENT TRANSMIT$COUNT}
    IF TRANSMIT$COUNT=0FFH THEN
    DO;
      CALL TRANSMIT;
      {CLEAR TRANSMIT$COUNT}
    END;
  END;
ELSE
DO;
  {TURN THE CURSER ON DURING THE AUTO REPEAT FUNCTION}
  IF TRANSMIT$TOGGLE = 1 THEN
  DO;
    CALL TRANSMIT;
    {COMPLEMENT TRANSMIT$TOGGLE}
  END;
END;
END AUTO$REPEAT;

/* PROCEDURE TRANSMIT- ONCE THE HOST COMPUTER SIGNALS THE 8051H BY BRINGING
THE CLEAR-TO-SEND LINE LOW, THE ASCII CHARACTER IS PUT INTO THE SERIAL PORT.*/

TRANSMIT:
PROCEDURE;
IF {THE TERMINAL IS ON-LINE} THEN
DO;
  {WAIT UNTIL THE CLEAR$TO$SEND LINE IS LOW AND UNTIL THE 8051 SERIAL PORT TX IS NOT BUSY (TRANSMIT$INT=1)}
  {TRANSMIT THE ASCII CODE}
  {CLEAR THE FLAG "TRANSMIT$INT". THE SERIAL PORT SERVICE ROUTINE WILL SET THE FLAG
  WHEN THE SERIAL PORT IS FINISHED TRANSMITTING}
END;
ELSE {THE TERMINAL IS IN THE LOCAL MODE}
DO;
  {PUT THE ASCII CODE IN THE FIFO}
  {INCREMENT THE FIFO POINTER}
  {SET SERIAL$INT}
END;
END TRANSMIT;

```

```

/*      PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER'S MESSAGES AND DETERMINES
WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE
THE PROCEDURE THEN ACTS ACCORDINGLY */

```

```

DECIPHER:
START$DECIPHER:

```

```

VALID$RECEPTION=0;
DO WHILE {THE FIFO IS NOT EMPTY AND THE CHARACTER IS DISPLAYABLE}
    RECEIVE={ASCII CODE}
    CALL DISPLAY;
    {NEXT CHARACTER}
END;

```

```

IF {CHARACTERS WERE DISPLAYED} THEN
    {DISABLE SERIAL PORT INTERRUPT}
    {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
    {ENABLE SERIAL PORT INTERRUPT}
    {SET THE VALID$RECEPTION FLAG}

```

```

IF {THE FIFO IS EMPTY} THEN {CLEAR THE "SERIAL$INT FLAG AND RETURN}

```

```

IF {THE NEXT CHARACTER IS AN "ESC" CODE } THEN
DO:

```

```

    {LOOK AT THE CHARACTER IN THE FIFO AFTER THE ESC CODE AND CALL THE CORRECT SUBROUTINE}

```

```

    ;
    CALL UP$CURSER;                /* ESC A */
    CALL DOWN$CURSER;              /* ESC B */
    CALL RIGHT$CURSER;             /* ESC C */
    CALL LEFT$CURSER;              /* ESC D */
    CALL CLEAR$SCREEN;              /* ESC E */
    CALL MOV$CURSER;                /* ESC F */
    ;
    CALL HOME;                     /* ESC H */
    ;
    CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN; /* ESC J */
    CALL BLINK;                     /* ESC K */

```

```

    {DISABLE THE SERIAL PORT INTERRUPT}
    {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
    {ENABLE THE SERIAL PORT INTERRUPT}
    {SET THE "VALID$RECEPTION" FLAG}

```

```

    IF {THE FIFO IS EMPTY} THEN {CLEAR THE SERIAL$INT FLAG AND RETURN}
END;

```



```
IF {THE NEXT CHARACTER IS A CONTROL CODE} THEN
DO;
```

```
{CALL THE RIGHT SUBROUTINE}
```

```
CALL LEFT$CURSER;
```

```
;
```

```
CALL LINE$FEED;
```

```
;
```

```
CALL CLEAR$SCREEN;
```

```
CALL CARRIAGE$RETURN;
```

```
{DISABLE THE SERIAL PORT INTERRUPT}
{MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
{ENABLE THE SERIAL PORT INTERRUPT}
{SET THE "VALID$RECEPTION" FLAG}
```

```
END;
```

```
IF {NO VALID CODE WAS RECEIVED ("VALID$RECEPTION" IS 0)} THEN
{THROW THE CHARACTER OUT AND MOVE THE REMAINING CONTENTS OF THE FIFO}
{UP TO THE BEGINNING}
```

```
IF {THE FIFO IS EMPTY} THEN {CLEAR THE SERIAL$INT FLAG AND RETURN}.
```

```
END DECIPHER;
```

```
/* PROCEDURE DISPLAY: THIS PROCEDURE WILL TAKE THE BYTE IN RAM LABELED
RECEIVE AND PUT IT INTO THE DISPLAY RAM. */
```

```
DISPLAY:
```

```
{PUT INTO THE DISPLAY RAM LOCATION POINTED TO BY "DISPLAY$RAM$POINTER"
THE CONTENTS OF RECEIVE}
```

```
IF {THE END OF THE DISPLAY MEMORY HAS BEEN REACHED} THEN
{RESET "DISPLAY$RAM$POINTER" TO THE BEGINNING OF THE RAM}
```

```
ELSE
{INCREMENT "DISPLAY$RAM$POINTER"}
```

```
IF {THE CURSER IS IN THE LAST COLUMN OF THE CRT DISPLAY} THEN
```

```
DO;
```

```
{MOVE THE CURSER BACK TO THE BEGINNING OF THE LINE}
```

```
IF {THE NEW DISPLAY RAM LOCATION HAS A END-OF-LINE CHARACTER IN IT} THEN
CALL FILL;
```

```
IF {THE CURSER IS ON THE LAST LINE OF THE CRT DISPLAY} THEN
```

```
CALL SCROLL;
```

```
ELSE
```

```
{MOVE THE CURSER TO THE NEXT LINE}
```

```
END;
```

```
ELSE
```

```
{INCREMENT THE CURSER TO THE NEXT LOCATION}
```

```
{TURN THE CURSER ON }
```

```
CALL LOADCURSER;
```

```
END DISPLAY;
```

```
/*      PROCEDURE LINE$FEED      */
```

```
LINE$FEED:
```

```
IF {THE CURSER IS IN THE LAST LINE OF THE CRT DISPLAY} THEN
```

```
CALL SCROLL;
```

```
ELSE
```

```
DO;
```

```
{MOVE THE CURSER TO THE NEXT LINE}
```

```
{TURN THE CURSER ON}
```

```
CALL LOAD$CURSER;
```

```
END;
```

```
IF {THE DISPLAY$RAM$POINTER IS ON THE LAST LINE IN THE DISPLAY RAM} THEN
```

```
{MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY RAM}
```

```
ELSE
```

```
{MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY RAM}
```

```
IF {THE FIRST CHARACTER IN THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER} THEN
```

```
CALL FILL;
```

```
END LINE$FEED;
```

```
/*      PROCEDURE SCROLL      */
```

```
SCROLL:
```

```
CALL BLANK;
```

```
{DISABLE VERTICAL RETRACE INTERRUPT}
```

```
IF {THE FIRST LINE OF THE CRT CONTAINS THE LAST LINE OF THE DISPLAY MEMORY} THEN
```

```
{MOVE THE POINTER "LINE0" TO THE BEGINNING OF THE DISPLAY MEMORY}
```

```
ELSE
```

```
{MOVE "LINE0" TO THE NEXT LINE IN THE DISPLAY MEMORY}
```

```
{ENABLE VERTICAL RETRACE INTERRUPT}
```

```
END SCROLL;
```

```
/*      PROCEDURE CLEAR SCREEN      */
```

```
CLEAR$SCREEN:
```

```
CALL HOME;
```

```
CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
```

```
END CLEAR$SCREEN;
```

```

/*      PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSER TO THE 0,0 POSITION */
HOME:
{MOVE THE CURSER POSITION TO THE UPPER LEFT HAND CORNER OF THE CRT}
{TURN THE CURSER ON}
CALL LOAD$CURSER;
{MOVE THE DISPLAY$RAM$POINTER TO THE CORRECT LOCATION IN THE DISPLAY RAM}
END HOME;

/*      PROCEDURE ERASE FROM CURSER TO END OF SCREEN: */
ERASE$FROM$CURSER$TO$END$OF$SCREEN:
CALL BLINE;
/* ERASE CURRENT LINE */
IF {THE CURSER IS NOT ON THE LAST LINE OF THE CRT DISPLAY} THEN
    STARTING WITH THE NEXT LINE,PUT AN END-OF-LINE CHARACTER (0F1H)
    IN THE DISPLAY RAM LOCATIONS THAT CORRESPOND TO THE BEGINNING OF
    THE CRT DISPLAY LINES UNTIL THE BOTTOM OF THE CRT SCREEN HAS BEEN REACHED}
END;
END ERASE$FROM$CURSER$TO$END$OF$SCREEN;

/*PROCEDURE MOV$CURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR
IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL
READ THE NEXT TWO BYTE TO DETERMINE WHERE TO MOVE THE CURSER. THE FIRST BYTE
IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION */
MOV$CURSER:
{WAIT UNTIL THE FIFO HAS RECEIVED THE NEXT TWO CHARACTERS}
{MOVE THE CURSER TO THE LOCATION SPECIFIED IN THE ESCAPE SEQUENCE}
{MOVE THE DISPLAY$RAM$POINTER TO THE CORRECT LOCATION}
IF THE FIRST CHARACTER IN THE NEW LINE HAS AN END-OF-LINE CHARACTER} THEN
    CALL FILL;
END;
{DISABLE THE SERIAL PORT INTERRUPT}
{MOVE THE REMAIN CONTENTS OF THE FIFO UP TWO LOCATIONS IN MEMORY}
{DECREMENT THE FIFO BY TWO}
{ENABLE THE SERIAL PORT INTERRUPT}
END MOV$CURSER;

/*      PROCEDURE LEFT CURSER: THIS PROCEDURE MOVES THE CURSER LEFT ONE COLUMN
BY SUBTRACTING 1 OF THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */
LEFT$CURSER:
IF {THE CURSER IS NOT IN THE FIRST LOCATION OF A LINE} THEN
DO;
    {MOVE THE CURSER LEFT BY ONE LOCATION}
    {TURN THE CURSER ON}
    CALL LOAD$CURSER;
    {DECREMENT THE DISPLAY$RAM$POINTER BY ONE}
END;
END LEFT$CURSER;

```

```

/*      PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
      BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

```

```

RIGHT$CURSER:

```

```

IF {THE CURSER IS NOT IN THE LAST POSITION OF THE CRT LINE} THEN

```

```

DO;

```

```

    {MOVE THE CURSER RIGHT BY ONE LOCATION}
    {TURN THE CURSER ON}
    CALL LOAD$CURSER;
    {INCREMENT THE DISPLAY$RAM$POINTER BY ONE}

```

```

END;

```

```

END RIGHT$CURSER;

```

```

/*      PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
      BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

```

UP$CURSER:

```

```

IF {THE CURSER IS NOT ON THE FIRST LINE OF THE CRT DISPLAY} THEN

```

```

DO;

```

```

    {MOVE THE CURSER UP ONE LINE}
    {TURN ON THE CURSER}
    CALL LOAD$CURSER;

```

```

    IF {THE DISPLAY$RAM$POINTER IS IN THE FIRST LINE OF DISPLAY MEMORY} THEN
        {MOVE THE DISPLAY$RAM$POINTER TO THE LAST LINE OF DISPLAY MEMORY}

```

```

    ELSE

```

```

        {MOVE THE DISPLAY$RAM$POINTER UP ONE LINE IN DISPLAY MEMORY}

```

```

    IF {THE FIRST LOCATION OF THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER} THEN

```

```

        CALL FILL;

```

```

END;

```

```

END UP$CURSER;

```

```

/*      PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
      BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

```

DOWN$CURSER:

```

```

IF {THE CURSER IS NOT ON THE LAST LINE OF THE CRT DISPLAY} THEN

```

```

DO;

```

```

    {TURN THE CURSER ON}
    {MOVE THE CURSER TO THE NEXT LINE}
    CALL LOAD$CURSER;

```

```

    IF {THE DISPLAY$RAM$POINTER IS NOT ON THE LAST LINE OF THE DISPLAY MEMORY} THEN
        {MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY MEMORY}

```

```

    ELSE

```

```

        {MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY MEMORY}

```

```

    IF {THE FIRST CHARACTER IN THE NEW LINE IS AN END-OF-LINE CHARACTER} THEN

```

```

        CALL FILL;

```

```

END;

```

```

END DOWN$CURSER;

```



```

/*      PROCEDURE CARRIAGE$RETURN      */
CARRIAGE$RETURN:
{MOVE THE DISPLAY$RAM$POINTER TO THE BEGINNING OF THE CURRENT LINE IN THE DISPLAY MEMORY}
{MOVE THE CURSER TO THE BEGINNING OF THE CURRENT LINE OF THE CRT DISPLAY}
{TURN THE CURSER ON}
CALL LOAD$CURSER;

END CARRIAGE$RETURN;

/*      PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND
LOADS IT INTO THE 8276 CURSER REGISTER.  */
LOAD$CURSER:
PROCEDURE;
IF {THE CURSER IS ON} THEN
{MOVE THE CURSER BACK ONTO THE CRT DISPLAY}
{DISABLE BUFFER INTERRUPT}
WRITE TO THE 8276 CURSER REGISTERS THE X,Y LOCATIONS}
{ENABLE BUFFER INTERRUPT}

END LOAD$CURSER;

/*      PROCEDURE CHECK BAUD RATE: THIS PROCEDURE READS THE THREE PORT PINS ON P1 AND SETS UP
THE SERIAL PORT FOR THE SPECIFIED BAUD RATE */
CHECK$BAUD$RATE:
{SET TIMER 1 TO MODE 1 AND AUTO RELOAD}
{TURN TIMER ON}
{ENABLE SERIAL PORT INTERRUPT}
{READ BAUD RATE SWITCHES AND SET UP RELOAD VALUE}

;
TH1=040H;      /* 00 IS NOT ALLOWED */
TH1=0A0H;      /* 150 BAUD */
TH1=0D0H;      /* 300 BAUD */
TH1=0E8H;      /* 600 BAUD */
TH1=0F4H;      /* 1200 BAUD */
TH1=0FAH;      /* 2400 BAUD */
TH1=0FDH;      /* 4800 BAUD */
TH1=0FDH;      /* 9600 BAUD */

END CHECK$BAUD$RATE;

```

```

/*      PROCEDURE READER: THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. THE
EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN
LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL
MEMORY ARRAY CALLED CURRENT$KEY */

```

READER:

```

{INITIALIZE FLAGS "KEY0"=0, "SAME"=1, 0 COUNTER=0}

DO UNTIL {ALL 8 KEYBOARD SCAN LINES ARE READ}
  {READ KEYBOARD SCAN}
  IF {NO KEY WAS PRESSED} THEN
    {INCREMENT 0 COUNTER}
  ELSE
    IF {THE KEY PRESSED WAS NOT THE SAME KEY THAT WAS PRESSED THE LAST TIME
      THE KEYBOARD WAS READ} THEN
      {CLEAR "SAME" AND WRITE NEW SCAN RESULT TO CURRENT$KEY RAM ARRAY}
    END;
  IF {ALL 8 SCANS DIDN'T HAVE A KEY PRESSED (0 COUNTER=8)} THEN
    {SET KEY0, AND CLEAR SAME}
  END READER;

```

```

/*      PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 WITH SPACES (20H ASCII)
DURING THE SCROLL ROUTINES.*/

```

```

BLANK:
DO I= {BEGINNING OF THE CRT DISPLAY (LINE0)} TO {LINE0 + 50H}
  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
NEXT I
END;

END BLANK;

```

```

/*      PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSOR TO THE END OF
THE DISPLAY LINE */

```

```

BLINE:
DO I= {CURRENT CURSOR POSITION ON CRT DISPLAY} TO {END OF ROW}
  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
NEXT I
END;

END BLINE;

```

```

/*      PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS A DISPLAY LINE WITH SPACES*/

```

```

FILL:
DO I= {BEGINNING OF THE LINE THAT THE CURSER IS ON} TO {END OF THE ROW}
  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
NEXT I
END;

END FILL;

```

BYE REGISTER

ADDRESS 1000H TO 17CFH

ADDRESS 1800H TO 1FCFH

ADDRESS 0001H (H08) TA (E7E) 09

ADDRESS 0000H (0000) TA 0000 19

ADDRESS 0001H (H040) TA 3778 29

DD (USING MISC):

```
D (USING MOVX):      00000000  00000000  00000000  00000000
```

```

$EJECT

CRT$CONTROLLER:
1 1 DO;

/***** DECLARE LITERALS *****/

2 1 DECLARE LIC LITERALLY 'LOCAL$LINE$CHANGE';
3 1 DECLARE REG LITERALLY 'REGISTER';
4 1 DECLARE CURRENT$KEY LITERALLY 'CURKEY';
5 1 DECLARE SERIAL$SERVICE LITERALLY 'SERBUF';
6 1 DECLARE DISPLAY$RAM$POINTER LITERALLY 'POINT';
7 1 DECLARE SERIAL$INT LITERALLY 'SERINT';
8 1 DECLARE TRANSMIT$INT LITERALLY 'TRNINT';
9 1 DECLARE CURSER$COLUMN LITERALLY 'CURSER';
10 1 DECLARE LAST$KEY LITERALLY 'LSTKEY';
11 1 DECLARE CURSER$COUNT LITERALLY 'COUNT';
12 1 DECLARE SCAN$INT LITERALLY 'SCAN';

/***** REGISTER DECLARATIONS FOR THE 8051 *****/

/***** BYTE REGISTERS *****/

13 1 DECLARE
    P0 BYTE AT (80H) REG,
    P1 BYTE AT (90H) REG,
    P2 BYTE AT (0A0H) REG,
    P3 BYTE AT (0B0H) REG,
    PSW BYTE AT (0D0H) REG,
    ACC BYTE AT (0E0H) REG,
    B BYTE AT (0F0H) REG,
    SP BYTE AT (81H) REG,
    DPL BYTE AT (82H) REG,
    DPH BYTE AT (83H) REG,
    PCON BYTE AT (87H) REG,
    TCON BYTE AT (88H) REG,
    TMOD BYTE AT (89H) REG,
    TL0 BYTE AT (8AH) REG,
    TL1 BYTE AT (8BH) REG,
    TH0 BYTE AT (8CH) REG,
    TH1 BYTE AT (8DH) REG,
    IE BYTE AT (0A8H) REG,
    IP BYTE AT (0B8H) REG,
    SCON BYTE AT (98H) REG,
    SBUF BYTE AT (99H) REG;

```


PL/M-51 COMPILER CRTCONTROLLER

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT
/***** BIT REGISTERS *****/
/***** PSW BITS *****/
14 1 DECLARE
CY BIT AT(0D7H) REG,
AC BIT AT(0D6H) REG,
FO BIT AT(0D5H) REG,
RS1 BIT AT(0D4H) REG,
RS0 BIT AT(0D3H) REG,
OV BIT AT(0D2H) REG,
P BIT AT(0D0H) REG,

/***** TOON BITS *****/
TF1 BIT AT(8FH) REG,
TR1 BIT AT(8EH) REG,
TF0 BIT AT(8DH) REG,
TR0 BIT AT(8CH) REG,
IE1 BIT AT(8BH) REG,
IT1 BIT AT(8AH) REG,
IE0 BIT AT(89H) REG,
IT0 BIT AT(88H) REG,

/***** IE BITS *****/
EA BIT AT(0AFH) REG,
ES BIT AT(0ACH) REG,
ET1 BIT AT(0ABH) REG,
EK1 BIT AT(0AAH) REG,
ET0 BIT AT(0A9H) REG,
EK0 BIT AT(0A8H) REG,

/***** IP BITS *****/
PS BIT AT(0BCH) REG,
PT1 BIT AT(0BBH) REG,
PK1 BIT AT(0BAH) REG,
PT0 BIT AT(0B9H) REG,
PK0 BIT AT(0B8H) REG,

/***** P3 BITS *****/
RD BIT AT(0B7H) REG,
WR BIT AT(0B6H) REG,
T1 BIT AT(0B5H) REG,
TO BIT AT(0B4H) REG,
INT1 BIT AT(0B3H) REG,
INT0 BIT AT(0B2H) REG,
TXD BIT AT(0B1H) REG,
RXD BIT AT(0B0H) REG,

/***** SOON BITS *****/
SM0 BIT AT(9FH) REG,
SM1 BIT AT(9EH) REG,
SM2 BIT AT(9DH) REG,
REN BIT AT(9CH) REG,
TB8 BIT AT(9BH) REG,
RB8 BIT AT(9AH) REG,
TI BIT AT(99H) REG,
RI BIT AT(98H) REG;

```

PL/M-51 COMPILER CRTCONTROLLER

RECEIVED 12-14-71

```

$EJECT
$IF SW1
/***** DECLARE CONSTANTS *****/
15 1 DECLARE LOW$SCAN(16) STRUCTURE
      (KEY(8) BYTE) CONSTANT
      ('890-',5CH,5EH,08H,00H,
/* SCAN 0, SHIFT KEY =0; 8,9,0,-,\,^, BACK SPACE */
      'uiop',5BH,'@',0AH,7FH,
/* SCAN 1, SHIFT =0; u,i,o,p,[,@, LINE FEED, DELETE */
      'jkl;:',00H,0DH,'7',
/* SCAN 2, SHIFT =0; j,k,l;,:, RETURN, 7 */
      'm',2CH,'.',00H,'/',00H,00H,00H,
/* SCAN 3, SHIFT =0; m,comma,.,/ */
      00H,'azxcvbn',
/* SCAN 4, SHIFT =0; a,z,x,c,v,b,n */
      'y',00H,00H,'dfgh',
/* SCAN 5, SHIFT =0; y, SPACE, d,f,g,h */
      09H,'qwsert',00H,
/* SCAN 6, SHIFT =0; TAB,q,w,s,e,r,t */
      1BH,'123456',00H,
/* SCAN 7, SHIFT =0;ESC,1,2,3,4,5,6 */
      28H,29H,00H,'=',7CH,7EH,08H,00H,
/* SCAN 0, SHIFT =1; (,),=,~, BACK SPACE */
      'UIOP',00H,00H,0AH,7FH,
/* SCAN 1, SHIFT =1; U,I,O,P, LINE FEED, DELETE */
      'JKL+*',00H,0DH,27H,
/* SCAN 2, SHIFT =1; J,K,L,+,*, RETURN, ' */
      'M<>',00H,3FH,00H,00H,00H,
/* SCAN 3, SHIFT =1; M,<,>,* */
      00H,'AZXCVEN',
/* SCAN 4, SHIFT =1; A,Z,X,C,V,B,N */
      'Y',00H,00H,'DFGH',
/* SCAN 5, SHIFT =1; Y, SPACE, D,F,G,H */
      09H,'QWERT',00H,
/* SCAN 6, SHIFT =1; TAB, Q,W,S,E,R,T */
      1BH,'!#$%&',00H);
/* SCAN 7, SHIFT =1;ESC,!,",#,$,%,& */
$ENDIF

```

PL/M-51 COMPILER CRICONTROLLER

RELINQUISHED TO THE PUBLIC DOMAIN

```

$EJECT
/*****DECLARE VARIABLES*****/

16 1  DECLARE
    $IF SW2
INPUT    BIT AT (0B4H)  REG,
$ENDIF
$IF SW1
    CAP$LOCK          BIT AT (095H)  REG,
    SHIFT$KEY         BIT AT (096H)  REG,
    CONTROL$KEY       BIT AT (097H)  REG,
$ENDIF
    LOCAL$LINE        BIT AT (0B5H)  REG,
    CLEAR$TO$SEND     BIT AT (093H)  REG,
    DATA$TERMINAL$READY BIT AT (094H) REG;

17 1  DECLARE (
    $IF SW1
        SAME,
        VALID$KEY,
        KEY0,
        LAST$SHIFT$KEY,
        LAST$CONTROL$KEY,
        LAST$CAP$LOCK,
    $ENDIF
    $IF SW2
        RCVF$G,
        SYNC,
        BY$FIN,
        KBD$INT,
        ERROR,
    $ENDIF
        NEWS$KEY,
        TRANSMIT$TOGGLE,
        CURSER$ON,
        SERIAL$INT,
        SCAN$INT,
        TRANSMIT$INT,
        ESC$SEQ,
        VALID$RECEPTION,
        LLC,
        ENSP)    BIT PUBLIC;

```

PL/M-51 COMPILER CRTCONTROLLER

```

18 1 DECLARE (
    I,
    J,
    K,
    ASCII$KEY,
    TRANSMIT$COUNT,
    TEMP,
    SHIFT,
    CURSER$COL,
    CURSER$COLUMN,
    CURSER$ROW,
    CURSER$COUNT,
    FIFO,
    RECEIVE)
    BYTE PUBLIC;

19 1 $IF SW1
    DECLARE LAST$KEY (8) BYTE PUBLIC;
    $ENDIF

    $IF SW2
    DECLARE LAST$KEY (2) BYTE PUBLIC;
    $ENDIF

20 1 DECLARE SERIAL (16) BYTE PUBLIC;

21 1 DECLARE DISPLAY$RAM (7CFH) BYTE AT (1000H) AUXILIARY;

22 1 DECLARE
    PARAMETER$ADDRESS BYTE AT (0000H) AUXILIARY,
    COMMAND$ADDRESS BYTE AT (0001H) AUXILIARY;

23 1 DECLARE (
    DISPLAY$RAM$POINTER,
    RASTER,
    LINE0,
    L)
    WORD PUBLIC;

```


PL/M-51 COMPILER CRTCONTROLLER

ELIOTM0010 3219M00-12-M,N

```

$EJECT                                     T0000000
/* PROCEDURE READER: THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. THE
EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN
LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL
MEMORY ARRAY CALLED CURRENT$KEY. THE PROCEDURE CONTROLS 2 STATUS FLAGS;
KEY0 AND SAME. KEY0 IS SET IF ALL 8 SCANS READ NO KEY WAS PRESSED.
IF ALL 8 SCANS ARE THE SAME AS THE LAST READING OF THE KEYBOARD, THEN
SAME IS SET. */
24 2  READER: PROCEDURE EXTERNAL;
25 1  END READER;

/* PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 SCAN WITH SPACES (20H ASCII)
DURING THE SCROLL ROUTINES.*/
26 2  BLANK: PROCEDURE EXTERNAL;
27 1  END BLANK;

/* PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSER TO THE END OF
THE DISPLAY LINE */
28 2  BLINE: PROCEDURE EXTERNAL;
29 1  END BLINE;

/* PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS THE CURSER LINE
WITH SPACES*/
30 1  FILL:
31 1  PROCEDURE EXTERNAL;
END FILL;

```

\$SELECT

```
/* PROCEDURE CHECK BAUD RATE: THIS PROCEDURE READS THE THREE PORT PINS ON P1 AND SETS UP
THE SERIAL PORT FOR THE SPECIFIED BAUD RATE */
```

```
32 1 CHECK$BAUD$RATE:
33 2 PROCEDURE;
34 2 TMOD=TMOD OR 20H;
35 2 TR1=1;
36 2 ES=1;
37 2 ENSP=1;
38 3 DO CASE (P1 AND 07H);
39 3 ;
40 3 TH1=040H;
41 3 TH1=0A0H;
42 3 TH1=0D0H;
43 3 TH1=0E8H;
44 3 TH1=0F4H;
45 3 TH1=0FAH;
46 3 TH1=0FDH;
47 3 END;
48 1 END CHECK$BAUD$RATE;
```

```
/* MODE 1
ENABLE RECEPTION*/
/* TIMER 1 AUTO RELOAD */
/* TIMER 1 ON */
/* ENABLE SERIAL INTERRUPT*/
/* SERIAL INTERRUPT MASK FLAG */
/* 00 IS NOT ALLOWED */
/* 150 BAUD */
/* 300 BAUD */
/* 600 BAUD */
/* 1200 BAUD */
/* 2400 BAUD */
/* 4800 BAUD */
/* 9600 BAUD */
```

```
/* PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND
LOADS IT INTO THE 8276 CURSER REGISTERS. */
```

```
49 1 LOAD$CURSER:
50 2 PROCEDURE;
51 2 IF CURSER$ON=1 THEN
52 2 CURSER$COL=CURSER$COLUMN;
53 2 EX1=0;
54 2 COMMAND$ADDRESS=80H;
55 2 PARAMETER$ADDRESS=CURSER$COL;
56 2 EX1=1;
57 1 END LOAD$CURSER;
```

```
/* DISABLE BUFFER INTERRUPT */
/* INITIALIZE CURSER COMMAND */
/* ENABLE BUFFER INTERRUPT */
```

```
/* PROCEDURE CARRIAGE$RETURN */
```

```
58 1 CARRIAGE$RETURN:
59 2 PROCEDURE;
60 2 DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-CURSER$COLUMN;
61 2 CURSER$COLUMN=0;
62 2 CURSER$ON=1;
63 1 CALL LOAD$CURSER;
END CARRIAGE$RETURN;
```

PL/M-51 COMPILER CRTCONTROLLER

PL/M-51 COMPILER CRTCONTROLLER

\$EJECT

TOBET

/* PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

64 1  DOWN$CURSER:
        PROCEDURE;
65 2  IF CURSER$ROW < 18H THEN
66 3      DO;
67 4      CURSER$ON=1;
68 5      CURSER$ROW=CURSER$ROW + 1;
69 6      CALL LOAD$CURSER;
70 7      IF DISPLAY$RAM$POINTER < 780H THEN
71 8          DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER + 50H;
72 9      ELSE
73 10         DISPLAY$RAM$POINTER=(DISPLAY$RAM$POINTER-780H);
74 11         L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
75 12         IF DISPLAY$RAM(L)=0F1H THEN
76 13             DO;
77 14                 CALL FILL;
78 15                 DISPLAY$RAM(L)=20H;
79 16             END;
80 17         END DOWN$CURSER;

```

/* PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

81 1  UP$CURSER:
        PROCEDURE;
82 2  IF CURSER$ROW > 0 THEN
83 3      DO;
84 4      CURSER$ROW=CURSER$ROW - 1;
85 5      CURSER$ON=1;
86 6      CALL LOAD$CURSER;
87 7      IF DISPLAY$RAM$POINTER<50H THEN
88 8          DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+780H;
89 9      ELSE
90 10         DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER - 50H;
91 11         L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
92 12         IF DISPLAY$RAM(L)=0F1H THEN
93 13             DO;
94 14                 CALL FILL;
95 15                 DISPLAY$RAM(L)=20H;
96 16             END;
97 17         END UP$CURSER;

```

PL/M-51 COMPILER CRTCONTROLLER

RELATIONSHIP RELATIONSHIP

\$EJECT

TABLE

/* PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

```

98 1  RIGHT$CURSER:
    PROCEDURE;
99 2  IF CURSER$COLUMN < 4FH THEN
100 3  DO;
101 3      CURSER$COLUMN=CURSER$COLUMN + 1;
102 3      CURSER$QN=1;
103 3      CALL LOAD$CURSER;
104 3      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER +1;
105 3  END;
106 1  END RIGHT$CURSER;

```

/* PROCEDURE LEFT CURSER: THIS PROCEDURE MOVES THE CURSER LEFT ONE COLUMN
BY SUBTRACTING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

```

107 1  LEFT$CURSER:
    PROCEDURE;
108 2  IF CURSER$COLUMN > 0 THEN
109 3  DO;
110 3      CURSER$COLUMN=CURSER$COLUMN - 1;
111 3      CURSER$QN=1;
112 3      CALL LOAD$CURSER;
113 3      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER -1;
114 3  END;
115 1  END LEFT$CURSER;

```


PL/M-51 COMPILER CRTCONTROLLER

SALVAGE/RECOVERED REPAIR/REPAIR

\$EJECT

THER

/* PROCEDURE MOV\$CURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR
IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL
READ THE NEXT TWO BYTE TO DETERMINE WHERE TO MOVE THE CURSER. THE FIRST BYTE
IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION */

```

116 1  MOV$CURSER:
PROCEDURE;
117 3  DO WHILE FIFO<4; /* WAIT UNTILL THE MOV$CURSER PARAMETERS*/
118 3  END; /* ARE RECEIVED INTO THE FIFO */
119 2  TEMP=CURSER$ROW;
120 2  CURSER$ROW=SERIAL(2);
121 2  IF CURSER$ROW>TEMP THEN
122 3  DO;
123 3  L=DISPLAY$RAM$POINTER+((CURSER$ROW-TEMP)*50H);
124 3  IF L>7CFH THEN /* IF OUT OF RAM RANGE */
125 3  DISPLAY$RAM$POINTER=L-7D0H; /* RAP AROUND TO BEGINNING */
126 3  ELSE /* OF RAM */
127 3  DISPLAY$RAM$POINTER=L;
128 2  END;
129 3  IF CURSER$ROW<TEMP THEN
130 4  DO;
131 4  L=(TEMP-CURSER$ROW)*50H;
132 4  IF DISPLAY$RAM$POINTER<L THEN /* IF OUT OF RAM RANGE*/
133 4  DISPLAY$RAM$POINTER=(7D0H-(L-DISPLAY$RAM$POINTER)); /* RAP AROUND TO END OF RAM*/
134 4  ELSE
135 4  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-L;
136 3  END;
137 2  TEMP=CURSER$COLUMN;
138 2  CURSER$COLUMN=SERIAL(3);
139 2  IF CURSER$COLUMN>TEMP THEN
140 2  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+(CURSER$COLUMN-TEMP);
141 2  ELSE
142 2  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-(TEMP-CURSER$COLUMN);
143 2  CURSER$ON=1;
144 2  CALL LOAD$CURSER;
145 2  L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
146 3  IF DISPLAY$RAM(L)=0F1H THEN /* LOOK FOR END FO LINE CHARACTER*/
147 3  CALL FILL; /* IF TRUE FILL WITH SPACES */
148 3  DISPLAY$RAM(L)=20H;
149 3  END;
150 2  ES=0;
151 3  DO I=2 TO FIFO-2;
152 3  SERIAL(I)=SERIAL(I+2);
153 3  END;
154 2  FIFO=FIFO-2;
155 2  ES=ENSP;
156 1  END MOV$CURSER;

```

```

SELECT
/*PROCEDURE ERASE FROM CURSER TO END OF SCREEN: */
157 1 ERASE$FROM$CURSER$TO$END$OF$SCREEN:
PROCEDURE;
158 2 CALL BLINE; /* ERASE CURRENT LINE */
159 2 IF CURSER$ROW < 18H THEN
160 3 DO;
161 3 L=DISPLAY$RAM$POINTER-CURSER$COLUMN+50H; /* GET NEXT LINE */
162 4 DO WHILE (L < 7D0H) AND (L <> (LINE0 AND 7FFH));
163 4 DISPLAY$RAM(L)=0F1H; /* ERASE UNTIL LINE0 OR */
164 4 L=L+50H; /* END OF DISPLAY RAM */
165 4 END;
166 3 L=0;
167 4 DO WHILE L <> (LINE0 AND 7FFH); /* ERASE UNTIL LINE0 */
168 4 DISPLAY$RAM(L)=0F1H;
169 4 L=L+50H;
170 4 END;
171 3 END;
172 1 END ERASE$FROM$CURSER$TO$END$OF$SCREEN;

/*PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSER TO THE 0,0 POSITION */
173 1 HOME:
PROCEDURE;
174 2 CURSER$ROW=00;
175 2 CURSER$COLUMN=00;
176 2 CURSER$ON=1;
177 2 CALL LOAD$CURSER;
178 2 DISPLAY$RAM$POINTER=(LINE0 AND 7FFH);
179 1 END HOME;

```

PL/M-51 COMPILER CRVCONTROLLER

PL/M-51 COMPILER CRVCONTROLLER

```

SELECT
/* PROCEDURE CLEAR SCREEN */
180 1  CLEAR$SCREEN:
      PROCEDURE;
181 2  CALL HOME;
182 2  CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
183 1  END CLEAR$SCREEN;

/* PROCEDURE SCROLL */
184 1  SCROLL:
      PROCEDURE;
185 2  CALL BLANK;
186 2  EX0=0;
187 2  IF LINE0= 1F80H THEN
188 2     LINE0= 1800H;
189 2  ELSE
      LINE0= LINE0+50H;
190 2  EX0=1;
191 1  END SCROLL;

/* PROCEDURE LINE$FEED */
192 1  LINE$FEED:
      PROCEDURE;
193 2  IF CURSER$ROW=18H THEN
194 2     CALL SCROLL;
195 2  ELSE
      DO;
196 3     CURSER$ROW= CURSER$ROW+1;
197 3     CURSER$ON=1;
198 3     CALL LOAD$CURSER;
199 3  END;
200 2  IF DISPLAY$RAM$POINTER > 77FH THEN
201 2     DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-780H;
202 2  ELSE
      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+50H;
203 2  L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
204 2  IF DISPLAY$RAM(L)=0F1H THEN
205 3  DO;
206 3     CALL FILL;
207 3     DISPLAY$RAM(L)=20H;
208 3  END;
209 1  END LINE$FEED;

```

PL/M-51 COMPILER CRTCONTROLLER

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT
/* PROCEDURE DISPLAY: THIS PROCEDURE WILL TAKE THE BYTE IN RAM LABELED
RECEIVE AND PUT IT INTO THE DISPLAY RAM. */

210 1  DISPLAY:
      PROCEDURE;
211 2  DISPLAY$RAM(DISPLAY$RAM$POINTER)=RECEIVE;
212 2  IF DISPLAY$RAM$POINTER=7CFH THEN /* IF END OF RAM */
213 2  DISPLAY$RAM$POINTER=000H; /* RAP AROUND TO BEGINNING */
214 2  ELSE
      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+1;
215 2  IF CURSER$COLUMN=4FH THEN
216 3  DO;
      CURSER$COLUMN=00H;
217 3  L=DISPLAY$RAM$POINTER;
218 3  IF DISPLAY$RAM(L)=0F1H THEN
219 3  DO;
220 4  CALL FILL;
221 4  DISPLAY$RAM(L)=20H;
222 4  END;
223 4  IF CURSER$ROW=18H THEN /* DISPLAY VERTICAL BETWEEN
224 3  CALL SCROLL; /* SHAVE VERTICAL BETWEEN JACOBIEN BRANE */
225 3  ELSE
226 3  CURSER$ROW=CURSER$ROW+1;
227 3  END;
228 2  ELSE
      CURSER$COLUMN=CURSER$COLUMN+1;
229 2  CURSER$ON=1;
230 2  CALL LOADCURSER;
231 1  END DISPLAY;

```


PL/M-51 COMPILER CRIOCONTROLLER

RECEIVED 12-14-81

SELECT

```

/* PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER'S MESSAGES AND DETERMINES
   WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE
   THE PROCEDURE THEN ACTS ACCORDINGLY */

```

```

232 1  DECIPHER:
233 2  PROCEDURE;
234 2  START$DECIPHER:
235 3  VALID$RECEPTION=0;
236 3  I=0;
237 3  DO WHILE (I<FIFO) AND (SERIAL(I)>1FH) AND (SERIAL(I)<7FH);
238 3  RECEIVE-SERIAL(I);
239 3  CALL DISPLAY;
240 3  I=I+1;
241 3  END;
242 2  IF I>0 THEN
243 3  DO;
244 3  ES=0; /* DISABLE SERIAL INTERRUPT WHILE MOVING FIFO */
245 3  K=FIFO-I;
246 3  DO J=0 TO K; /* MOVE FIFO */
247 4  SERIAL(J)=SERIAL(I);
248 4  I=I+1;
249 4  END;
250 3  FIFO=K;
251 3  ES=ENSP; /* ENABLE SERIAL INTERRUPT */
252 3  VALID$RECEPTION=1;
253 3  END;
254 2  IF FIFO=0 THEN
255 3  DO;
256 3  SERIAL$INT=0;
257 3  GOTO END$DECIPHER;
258 3  END;
259 2  IF (SERIAL(0)=1BH) THEN
260 3  DO;
261 3  IF (ESC$SEQ=1) AND (FIFO<2) THEN
262 3  GOTO END$DECIPHER;
263 3  K=(SERIAL(1) AND 5FH)-40H;
264 3  IF (K > 00H) AND (K < 0CH) THEN
265 4  DO;
266 5  DO CASE K;
267 5  ;
268 5  CALL UP$CURSER; /* ESC A */
269 5  CALL DOWN$CURSER; /* ESC B */
270 5  CALL RIGHT$CURSER; /* ESC C */
271 5  CALL LEFT$CURSER; /* ESC D */
272 5  CALL CLEAR$SCREEN; /* ESC E */
273 5  CALL MOV$CURSER; /* ESC F */
274 5  ;
275 5  CALL HOME; /* ESC H */
276 5  ;
277 5  CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN; /* ESC J */
278 5  CALL BLINK; /* ESC K */
279 5  END;
280 3  END;
281 2  ES=0; /* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO */

```

```

280 4      DO I=0 TO (FIFO-2);
281 4          SERIAL(I)=SERIAL(I+2); /* MOVE FIFO */
282 4      END;
283 3      FIFO=FIFO-2;
284 3      ES=ENSP; /* ENABLE SERIAL INTERRUPTS */
285 3      VALID$RECEPTION=1;
286 3      IF FIFO=0 THEN
287 4          DO;
288 4              SERIAL$INT=0;
289 4              GOTO END$DECIPHER;
290 4          END;
291 3      END;
292 2      IF (SERIAL(0) > 07H) AND (SERIAL(0) < 0EH) THEN
293 3      DO;
294 4          DO CASE (SERIAL(0) - 08H);
295 4              CALL LEFT$CURSER; /* CTL H */
296 4              ;
297 4              CALL LINE$FEED; /* CTL J */
298 4              ;
299 4              CALL CLEAR$SCREEN; /* CTL L */
300 4              CALL CARRIAGE$RETURN; /* CTL M */
301 4          END;
302 3      ES=0; /* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO */
303 4      DO I=0 TO (FIFO-1);
304 4          SERIAL(I)=SERIAL(I+1); /* MOVE FIFO */
305 4      END;
306 3      FIFO=FIFO-1;
307 3      ES=ENSP; /* ENABLE SERIAL INTERRUPTS */
308 3      VALID$RECEPTION=1;
309 3      END;
310 2      IF VALID$RECEPTION=0 THEN
311 3      DO;
312 3          ES=0;
313 4          DO I=0 TO (FIFO-1);
314 4              SERIAL(I)=SERIAL(I+1); /* IF CHARACTER IS UNRECOGNIZED THEN */
315 4          END; /* TRASH IT */
316 3          FIFO=FIFO-1;
317 3          ES=ENSP;
318 3      END;
319 2      IF FIFO=0 THEN
320 2          SERIAL$INT=0;
321 2      END$DECIPHER;
322 2      END DECIPHER;

```

PL/M-51 COMPILER CRTCONTROLLER

PL/M-51 COMPILER CRTCONTROLLER

\$EJECT

\$EJECT

/* PROCEDURE TRANSMIT- THIS PROCEDURE LOOKS AT THE CLEAR TO SEND PIN FOR AN ACTIVE LOW SIGNAL. ONCE THE MAIN COMPUTER SIGNALS THE 8051 THE ASCII CHARACTER IS PUT INTO THE SERIAL PORT.*/

```

322 1  TRANSMIT:
      PROCEDURE;
323 2  IF LOCAL$LINE =1 THEN
324 3  DO;
325 4      DO WHILE (CLEAR$TO$SEND=1) OR (TRANSMIT$INT=0);
326 4      END;
327 3      SBUF=ASCIIKEY;
328 3      TRANSMIT$INT=0;
329 3  END;
330 2  ELSE
      DO;
331 3      SERIAL(FIFO)=ASCIIKEY;
332 3      FIFO=FIFO+1;
333 3      SERIAL$INT=1;
334 3  END;
335 1  END TRANSMIT;

```

/* PROCEDURE AUTO\$REPEAT: THIS PROCEDURE WILL PERFORM AN AUTO REPEAT FUNCTION AFTER A FIXED DELAY PERIOD */

```

336 1  AUTO$REPEAT:
      PROCEDURE;
337 2  IF NEW$KEY=1 THEN
338 3  DO;
339 3      TRANSMIT$TOGGLE=0;
340 3      TRANSMIT$COUNT=00H;
341 3      CALL TRANSMIT;          /* FIRST CHARACTER */
342 3      NEW$KEY=0;
343 3  END;
344 2  ELSE
      DO;
345 3      IF TRANSMIT$COUNT <> 00H THEN
346 4      DO;
347 4          TRANSMIT$COUNT=TRANSMIT$COUNT+1;
348 4          IF TRANSMIT$COUNT=0FFH THEN /*DELAY BETWEEN FIRST CHARACTER AND THE SECOND */
349 5          DO;
350 5              CALL TRANSMIT;          /*SECOND CHARACTER */
351 5              TRANSMIT$COUNT=00;
352 5          END;
353 4      END;
354 3      ELSE
          DO;
355 4          CURSER$ON=1;
356 4          CURSER$COUNT=0;
357 4          IF TRANSMIT$TOGGLE = 1 THEN /* 2 VERT FRAMES BETWEEN 3RD TO NTH CHARACTER */
358 4              CALL TRANSMIT;          /* 3RD THROUGH NTH CHARACTER */
359 4          TRANSMIT$TOGGLE= NOT TRANSMIT$TOGGLE;
360 4      END;
361 3  END;
362 1  END AUTO$REPEAT;

```

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT

/***** START MAIN PROGRAM *****/
/* BEGIN BY PUTTING ASCII CODE FOR BLANK IN THE DISPLAY RAM; */

363 1  INIT:
364 2  DO L=0 TO 7FH;
365 2  DISPLAY$RAM(L)=20H;

/* INITIALIZE POINTERS, RAM BITS, ETC. */

366 1  ESC$SEQ=0;
367 1  SCANS$INT=0;
368 1  SERIAL$INT=0;
369 1  FIFO=0;
370 1  CURSER$COUNT=0;
371 1  LLC=0;
372 1  DATA$TERMINAL$READY=1;
373 1  TOON=05H;
374 1  LINE0=1800H;
375 1  RASTER=1800H;
376 1  DISPLAY$RAM$POINTER=0000H;
377 1  TRANSMIT$INT=1;

$IF SW1

378 2  DO I=0 TO 7;
379 2  LAST$KEY(I)=00H;
380 2  END;

381 1  VALID$KEY=0;
382 1  LAST$SHIFT$KEY=1;
383 1  LAST$CONTROL$KEY=1;
384 1  LAST$CAP$LOCK=1;
$ENDIF

$IF SW2
RCVFLG=0;
SYNC=0;
BYFIN=0;
KBDINT=0;
ERROR=0;
$ENDIF

/* INITIALIZE THE 8276 */

385 1  COMMAND$ADDRESS=00H; /* RESET THE 8276 */
386 1  PARAMETER$ADDRESS=4FH; /* NORMAL ROWS, 80 CHARACTER/ROW */
387 1  PARAMETER$ADDRESS=58H; /* 2 ROW COUNTS PER VERTICAL RETRACE
25 ROWS PER FRAME */
388 1  PARAMETER$ADDRESS=89H; /* LINE 9 IS THE UNDERLINE POSITION
10 LINES PER ROW */
389 1  PARAMETER$ADDRESS=0F9H; /* OFFSET LINE COUNTER, NON-TRANSPARENT FIELD ATTRIBUTE

```


PL/M-51 COMPILER CRTCONTROLLER

NON-BLINKING UNDERLINE CURSER, 20 CHARACTER COUNTS PER
HORIZONTAL RETRACE */

```

390 1      TEMP=COMMAND$ADDRESS;
391 1      CURSER$COLUMN=00H;
392 1      CURSER$ROW=00H;
393 1      CURSER$COL=00H;
394 1      CALL LOAD$CURSER;
395 1      TEMP=COMMAND$ADDRESS;

396 1      COMMAND$ADDRESS=0E0H;          /* PRESET 8276 COUNTERS */
397 1      TEMP=COMMAND$ADDRESS;

398 1      COMMAND$ADDRESS=23H;          /* START DISPLAY */
399 1      COMMAND$ADDRESS=0A0H;          /* ENABLE INTERRUPTS */
400 1      TEMP=COMMAND$ADDRESS;

      /* SET UP INTERRUPTS AND PRIORITIES */

      $IF SW1
401 1      IP=10H;                      /* SERIAL PORT HAS HIGHEST PRIORITY */
402 1      IE=85H;                      /* ENABLE 8051 EXTERNAL INTERRUPTS */
      $ENDIF

      $IF SW2
      IP=10H;                      /* SERIAL PORT HAS HIGHEST PRIORITY */
      IE=87H;                      /* ENABLE TIMER0 INTERRUPT */
      TMOD=05H;                    /* TIMER 0 =EVENT COUNTER */
      TL0=0FFH;
      TH0=0FFH;                    /* INITIALIZE COUNTER TO FFFFH */
      TR0=1;
      $ENDIF

/* PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A
SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT
WILL BE TRANSMITTED TO THE HOST COMPUTER.*/

403 1      SCANNER:
404 1      EA=1;
405 1      DATA$TERMINAL$READY=0;
406 2      IF CURSER$COUNT=1FH THEN    /* PROGRAMMABLE CURSER BLINK */
407 2      DO;
408 2          CURSER$ON=NOT CURSER$ON;
409 2          CURSER$COUNT=00;
410 2          IF CURSER$ON=0 THEN
411 2              CURSER$COL=7FH;
412 2              CALL LOAD$CURSER;
413 2          END;
414 2          IF LLC<>LOCAL$LINE THEN    /* IF LOCAL/LINE HAS CHANGED STATUS */
415 2          DO;
416 2              IF LOCAL$LINE=0 THEN
417 2                  ENSP=0;
418 2                  ES=0;
419 2              ELSE
420 2                  CALL CHECK$BAUD$RATE;
421 2                  LLC=LOCAL$LINE;
422 2              END;
423 2          $IF SW1
424 2          DO WHILE SCAN$INT=0;        /* WAIT UNTIL VERTICAL RETRACE BEFORE */
425 2              IF SERIAL$INT=1 THEN    /* SCANNING THE KEYBOARD */
426 2              CALL DECIPHER;
426 2          END;

```

```

SEJECT
427 1 CALL READER;
428 1 IF VALID$KEY = 1 AND SAME=1 AND (LAST$SHIFT$KEY=SHIFT$KEY) AND
    (LAST$CAP$LOCK=CAP$LOCK) AND (LAST$CONTROL$KEY=CONTROL$KEY) THEN
429 1 CALL AUTO$REPEAT;
430 1 ELSE
    DO;
431 2 IF KEY0=0 AND SAME=0 THEN
432 3 DO;
433 3 TEMP =0;
434 3 K=0;
435 4 DO WHILE LAST$KEY (K)=0;
436 4 K=K+1;
437 4 END;
438 3 TEMP=LAST$KEY (K);
439 4 DO I=(K+1) TO 7;
440 4 TEMP=TEMP+LAST$KEY (I);
441 4 END;
442 3 IF TEMP=LAST$KEY (K) THEN
443 4 DO;
444 4 J=0;
445 5 DO WHILE (TEMP AND 01H)=0;
446 5 TEMP=SHR (TEMP,1);
447 5 J=J+1;
448 5 END;
449 4 IF TEMP >1 THEN
450 5 DO;
451 5 VALID$KEY=0;
452 5 NEWS$KEY=0;
453 5 END;
454 4 ELSE
455 5 DO;
456 5 IF CONTROL$KEY=0 THEN
457 5 ASCII$KEY=(LOW$SCAN (K).KEY (J)) AND 1FH;
458 6 DO;
459 6 IF SHIFT$KEY=0 THEN
460 6 ASCII$KEY=LOW$SCAN (K+08H).KEY (J);
461 7 ELSE
462 7 ASCII$KEY=LOW$SCAN (K).KEY (J);
463 7 IF (CAP$LOCK=0) AND (ASCII$KEY>60H) AND (ASCII$KEY<7BH) THEN
464 7 ASCII$KEY=ASCII$KEY-20H;
465 8 IF LLC=0 THEN
466 8 DO;
467 8 IF ASCII$KEY=1BH THEN
468 8 ESC$SEQ=1;
469 8 ELSE
470 8 ESC$SEQ=0;
471 8 END;
472 7 END;
473 6 LAST$SHIFT$KEY=SHIFT$KEY;
474 5 LAST$CAP$LOCK=CAP$LOCK;
475 5 LAST$CONTROL$KEY=CONTROL$KEY;
476 5 VALID$KEY=1;
477 5 NEWS$KEY=1;
478 5 END;
479 3 END;
480 4 VALID$KEY=0;
481 4 NEWS$KEY=0;
482 4 END;
483 3 END;
484 2 END;

```

SENDIF

PL/M-51 COMPILER CRICONTROLLER

HEATH 48304362A JPM 12-83M

\$EJECT

\$IF SW2

IF SERIAL\$INT=1 THEN

CALL DECIPHER;

IF KBDINT =1 THEN

DO;

IF ERROR =0 THEN

DO;

ASCII\$KEY=LAST\$KEY (1);

NEWSKEY=1;

CALL AUTOREPEAT;

KBDINT=0;

END;

ERROR=0;

KBDINT=0;

END;

\$ENDIF

485 1 GOTO SCANNER;

486 1 END;

MODULE INFORMATION:

(STATIC+OVERLAYABLE)

CODE SIZE

= 08E6H 2278D

CONSTANT SIZE

= 0080H 128D

DIRECT VARIABLE SIZE

= 2DH+00H 45D+ 0D

INDIRECT VARIABLE SIZE

= 00H+00H 0D+ 0D

BIT SIZE

= 10H+00H 16D+ 0D

BIT-ADDRESSABLE SIZE

= 00H+00H 0D+ 0D

AUXILIARY VARIABLE SIZE

= 0000H 0D

MAXIMUM STACK SIZE

= 000CH 12D

REGISTER-BANK (S) USED:

0

1056 LINES READ

0 PROGRAM ERROR(S)

END OF PL/M-51 COMPILATION

MCS-51 MACRO ASSEMBLER CNTASM

ISIS-II MCS-51 MACRO ASSEMBLER V2.1
 OBJECT MODULE PLACED IN :F1:CNTASM.OBJ
 ASSEMBLER INVOKED BY: ASM51 :F1:CR1ASM.SRC

```

LUC  OBJ      LINE SOURCE
      1
      2
      3
      4 ;
      5 ;
      6
      7      PUBLIC BLANK
      8      PUBLIC BLINE
      9      PUBLIC FILL
     10      EXTRN DATA (LINE0,MASTER,POINT,SERIAL,FIFO,CURSER,COUNT,L)
     11      EXTRN BIT (SERINT,ESCSEQ,TRNINT,SCAN)
     12
     13
     14      CSEG AT(03H)
0003 8020     15      SJMP      VENT      ;RESET RASTER TO LINE0 AND SCAN KEYBOARD
     16
     17 ;      EXTRA CODE (DETACH)
     18 ;      CSEG AT(0BH)
     19 ;      LJMP      DETACH      ;NEEDED IF DECODED KEYBOARD IS USED
     20
     21      CSEG AT(013H)
0013 802A     22      SJMP      BUFFER      ;FILL 8276 ROW BUFFER
     23
     24      CSEG AT(023H)
0023 802D     25      SJMP      SERBUF      ;STICK SERIAL INFORMATION INTO THE FIFO
     26
     27      CSEG
     28
0025 C000     29      VENT:  PUSH      PSH      ;PUSH REG USED BY PLM51
0027 C0E0     30      PUSH      ACC
0029 C000     31      PUSH      U0H
002B 850000   F 32      MOV      MASTER,LINE0      ;REINITIALIZE RASTER TO LINE0
002E 850000   F 33      MOV      MASTER+1,LINE0+1
0031 7801     34      MOV      R0,#01H      ;CLR 8276 INTERRUPT FLAG
0033 E2       35      MOVX     A,R0
0034 0500     36      INC      COUNT      ;INCR CURSER COUNT REGISTER
0036 0200     37      SETB     SCAN      ;FOR DEBOUNCE ROUTINE
0038 0000     38      POP      U0H      ;POP REGISTERS
003A 00E0     39      POP      ACC
003C 0000     40      POP      PSH
003E 52       41      RETI
     42
     43
003F C0D0     44      BUFFER: PUSH      PSH      ;PUSH ALL REG USED BY PLM51 CODE
0041 C0E0     45      PUSH      ACL
0043 C0B2     46      PUSH      DPL
0045 C0B3     47      PUSH      UPH
0047 11FA     48      ACALL    UMA      ;FILL 8276 ROW BUFFER
0049 00B3     49      POP      UPH      ;POP REGISTERS
004B 00B2     50      POP      DPL
004D 00E0     51      POP      ACL
004F 0000     52      POP      PSH
0051 32       53      RETI
     54
     55 +1  EJECT
  
```


MCS-51 MACRO ASSEMBLER CRTASM

```

LUC OBJ      LINE SOURCE
0052 309904   56 SERBUF: JNB 01099H,CVER ;IF TRANSMIT BIT NOT SET THEN CHECK RECEIVE
0053 C299     58 CLM 099H ;CLR TRANSMISSION INTERRUPT FLAG
0057 0200     59 SETB 1RINT ;SETB TRANS INT FOR PLM51 STATUS CHECK
0059 209828   60 OVER: JB 98H,CGBACK ;IF HI NOT SET GGBACK
005C C001     61 PUSH 01 ;READ SBUFF
005E A999     62 MOV 01,SBUFF ;CLEAR RI BIT
0060 C298     63 CLM 098H ;PUSH REGISTERS USED BY PLM51
0062 C000     64 PUSH PSW
0064 C0E0     65 PUSH ACC
0066 C000     66 PUSH 00H
0068 C200     67 CLR ESCSEQ ;CLR ESC SEQUENCE FLAG
006A 7400     68 MOV A,#SERIAL ;GET SERIAL FIFO RAM START LOCATION
006C 2500     69 ADD A,#FIFO ;AND FIND HOW FAR INTO THE FIFO WE ARE
006E F8       70 MOV R0,A ;PUT IT INTO R0
006F E9       71 MOV A,R1
0070 C2E7     72 CLM 0E7H ;CLR BIT 7 OF ACC
0072 F6       73 MOV 0RU,A ;PUT DATA IN FIFO
0073 641B02   74 CJNE A,#10H,OVER1 ;IF DATA IS NOT A ESC KEY THEN GO OVER
0076 0200     75 SETB ESCSEQ ;SET ESC SEQUENCE FLAG
0078 0500     76 OVER1: INC FIFC ;MOV FIFO TO NEXT LOCATION
007A 0200     77 SETB SERINT ;SET SERIAL INT BIT FOR PLM51 STATUS CHECK
007C 0000     78 POP 00H ;POP REGISTERS
007E 00E0     79 POP ACC
0080 0000     80 POP PSW
0082 0001     81 POP 01H
0084 32       82 GGBACK: RETI
0085 C000     83
0087 C0E0     84 BLANK: PUSH PSW ;PUSH REG USED BY PLM51
0089 C0E0     85 PUSH ACC
0089 C082     86 PUSH DPL
008B C0D3     87 PUSH DPH
008D C000     88 PUSH 00H
008F 850082   89 MOV DPL,LINE0+1 ;GET LINE0 INFO
0092 850063   90 MOV DPH,LINE0 ;AND PUT IT INTO DPTH
0095 7850     91 MOV R0,#50H ;NUMBER OF CHARACTERS IN A LINE
0097 7420     92 NOTYET: MOV 0A,#20H ;ASCII SPACE CHARACTER
0099 F0       93 MOVX 0CPT, A ;MOV TO DISPLAY RAM
009A A3       94 INC DPTH ;INCR TO NEXT DISPLAY RAM LOCATION
009B 08FA     95 UJNZ R0,NOTYET ;IF ALL 50H LOCATIONS ARE NOT FILLED
009D 0000     96 ;GO DO MORE
009D 0000     97 POP 00H ;POP REGISTERS
009F D063     98 POP 0PH
00A1 0062     99 POP DPL
00A3 00E0    100 POP ACC
00A5 0000    101 POP PSW
00A7 22      102 NET
00A7 22      103 +1 SEJECT

```

MUS-51 MACRO ASSEMBLER CRTASM

LUC	OBJ	LINE	SOURCE	
		104		
00A8	C000	105	BLINE: PUSH PSW	;PUSH REGISTERS USED BY PLM51
00AA	C0E0	106	PUSH ACC	
00AC	C0E2	107	PUSH UPL	
00AE	C0E3	108	PUSH UPH	
00B0	C000	109	PUSH UOH	
00B2	850083 F	110	MOV UPH,PCINT	;GET CURRENT DISPLAY RAM LOCATION
00B5	850082 F	111	MOV UPL,PCINT+1	
00B8	438310	112	URL UPH,#10H	;SET BIT 15 FOR RAM ADDRESS DECODING
00B8	A800 F	113	MOV R0,CURSER	;GET CURSER COLUMN INFO TO TELL HOW
		114		;FAR INTO THE ROW YOU ARE
00BD	1420	115	CONT1: MOV A,#20H	;ASCII SPACE CHARACTER
00BF	F0	116	MOVX 8DPTH,A	;MOV TO DISPLAY RAM
00C0	A3	117	INC UPIR	;INCH TO NEXT DISPLAY RAM LOCATION
00C1	08	118	INC R0	
00C2	8850F8	119	CJNE R0,#50H,CONT1	;IF NOT AT THE END OF THE LINE
		120		; CONTINUE
00C5	0000	121	POP UOH	;POP REGISTERS
00C7	0083	122	POP UPH	
00C9	0082	123	POP UPL	
00CB	00E0	124	POP ACC	
00CD	0000	125	POP PSW	
00CF	22	126	RET	
		127		
00D0	C000	128	FILL: PUSH PSW	;PUSH REGISTERS USED BY PLM51
00D2	C0E0	129	PUSH ACC	
00D4	C082	130	PUSH UPL	
00D6	C083	131	PUSH UPH	
00D8	C000	132	PUSH UOH	
00DA	C3	133	CLR C	
00DB	850083 F	134	MOV UPH,L	;GET BEGINNING OF LINE RAM LOCATION
00DE	850082 F	135	MOV UPL,L+1	;CALCULATED BY PLM51
00E1	438310	136	URL UPH,#10H	;SET BIT 15 FOR DISPLAY RAM ADDRESS DECODE
00E4	184F	137	MOV R0,#4FH	;SET UP COUNTER FOR 50H LOCATIONS
00E6	A3	138	INC UPIR	;GO PAST THE OF1H
00E7	7420	139	CONT2: MOV A,#20H	;ASCII SPACE CHARACTER
00E9	F0	140	MOVX 8DPTH,A	;MOVE TO DISPLAY RAM
00EA	A3	141	INC UPIR	;INCH TO NEXT DISPLAY RAM LOCATION
00EB	08FA	142	DJNZ R0,CUNT2	;IF ALL 79 LOCATIONS HAVE NOT BEEN FILLED
		143		;THEN CONTINUE
00ED	0000	144	POP UOH	;POP REGISTERS
00EF	0083	145	POP UPH	
00F1	0082	146	POP UPL	
00F3	00E0	147	POP ACC	
00F5	0000	148	POP PSW	
00F7	22	149	RET	
		150		
		151		
		152	+1 \$EJECT	

MCS-51 MACRO ASSEMBLER CNTASM

HEATND HJHMH32A 09JAN 12-83M

LUC	OBJ	LINE	SOURCE	OBJ	LINE	OBJ	LINE
		153					
		154	;				
		155	;THIS ROUTINE MOVES DISPLAY RAM DATA TO ROM BUFFER OF 8276				
		156	;				
		157					
00F0	21dB	158	UDONE: AJMP	UMADNE	XVDM		
		159					
00FA	850083 F	160	UMA: MOV	UPH,MASTER	XVDM	;LOAD XFER POINTER HIGH BYTE	
00FD	850082 F	161	MOV	UPL,MASTER+1	XVDM	;LOAD XFER POINTER LOW BYTE	
0100	E0	162	MOVX	A,ADPTM	XVDM		
0101	A3	163	INC	UPTR	XVDM		
0102	2003F3	164	JB	UB3H,UDONE	XVDM	;IF IN11 HIGH, THEN UMA IS OVER	
0105	E0	165	MOVX	A,ADPTM	XVDM		
0106	A3	166	INC	UPTR	XVDM		
0107	E0	167	MOVX	A,ADPTM	XVDM		
0108	A3	168	INC	UPTR	XVDM		
0109	E0	169	MOVX	A,ADPTM	XVDM		
010A	A3	170	INC	UPTR	XVDM		
010B	E0	171	MOVX	A,ADPTM	XVDM		
010C	A3	172	INC	UPTR	XVDM		
010D	E0	173	MOVX	A,ADPTM	XVDM		
010E	A3	174	INC	UPTR	XVDM		
010F	E0	175	MOVX	A,ADPTM	XVDM		
0110	A3	176	INC	UPTR	XVDM		
0111	E0	177	MOVX	A,ADPTM	XVDM		
0112	A3	178	INC	UPTR	XVDM		
0113	E0	179	MOVX	A,ADPTM	XVDM		
0114	A3	180	INC	UPTR	XVDM		
0115	E0	181	TEN: MOVX	A,ADPTM	XVDM		
0116	A3	182	INC	UPTR	XVDM		
0117	E0	183	MOVX	A,ADPTM	XVDM		
0118	A3	184	INC	UPTR	XVDM		
0119	E0	185	MOVX	A,ADPTM	XVDM		
011A	A3	186	INC	UPTR	XVDM		
011B	E0	187	MOVX	A,ADPTM	XVDM		
011C	A3	188	INC	UPTR	XVDM		
011D	E0	189	MOVX	A,ADPTM	XVDM		
011E	A3	190	INC	UPTR	XVDM		
011F	E0	191	MOVX	A,ADPTM	XVDM		
0120	A3	192	INC	UPTR	XVDM		
0121	E0	193	MOVX	A,ADPTM	XVDM		
0122	A3	194	INC	UPTR	XVDM		
0123	E0	195	MOVX	A,ADPTM	XVDM		
0124	A3	196	INC	UPTR	XVDM		
0125	E0	197	MOVX	A,ADPTM	XVDM		
0126	A3	198	INC	UPTR	XVDM		
0127	E0	199	MOVX	A,ADPTM	XVDM		
0128	A3	200	INC	UPTR	XVDM		
0129	E0	201	TWENTY: MOVX	A,ADPTM	XVDM		
012A	A3	202	INC	UPTR	XVDM		
012B	E0	203	MOVX	A,ADPTM	XVDM		
012C	A3	204	INC	UPTR	XVDM		
012D	E0	205	MOVX	A,ADPTM	XVDM		
012E	A3	206	INC	UPTR	XVDM		
012F	E0	207	MOVX	A,ADPTM	XVDM		

LUC	OBJ	LINE	SOURCE	TIME	DATE
0130	A3	208	INC UPTR	121	
0131	E0	209	MOVX A,CDPTH	121	
0132	A3	210	INC UPTR	121	
0133	E0	211	MOVX A,CDPTH	121	
0134	A3	212	INC UPTR	121	
0135	E0	213	MOVX A,CDPTH	121	
0136	A3	214	INC UPTR	121	
0137	E0	215	MOVX A,CDPTH	121	
0138	A3	216	INC UPTR	121	
0139	E0	217	MOVX A,CDPTH	121	
013A	A3	218	INC UPTR	121	
013B	E0	219	MOVX A,CDPTH	121	
013C	A3	220	INC UPTR	121	
013D	E0	221	THIRTY: MOVX A,CDPTH	121	
013E	A3	222	INC UPTR	121	
013F	E0	223	MOVX A,CDPTH	121	
0140	A3	224	INC UPTR	121	
0141	E0	225	MOVX A,CDPTH	121	
0142	A3	226	INC UPTR	121	
0143	E0	227	MOVX A,CDPTH	121	
0144	A3	228	INC UPTR	121	
0145	E0	229	MOVX A,CDPTH	121	
0146	A3	230	INC UPTR	121	
0147	E0	231	MOVX A,CDPTH	121	
0148	A3	232	INC UPTR	121	
0149	E0	233	MOVX A,CDPTH	121	
014A	A3	234	INC UPTR	121	
014B	E0	235	MOVX A,CDPTH	121	
014C	A3	236	INC UPTR	121	
014D	E0	237	MOVX A,CDPTH	121	
014E	A3	238	INC UPTR	121	
014F	E0	239	MOVX A,CDPTH	121	
0150	A3	240	INC UPTR	121	
0151	E0	241	FORTY: MOVX A,CDPTH	121	
0152	A3	242	INC UPTR	121	
0153	E0	243	MOVX A,CDPTH	121	
0154	A3	244	INC UPTR	121	
0155	E0	245	MOVX A,CDPTH	121	
0156	A3	246	INC UPTR	121	
0157	E0	247	MOVX A,CDPTH	121	
0158	A3	248	INC UPTR	121	
0159	E0	249	MOVX A,CDPTH	121	
015A	A3	250	INC UPTR	121	
015B	E0	251	MOVX A,CDPTH	121	
015C	A3	252	INC UPTR	121	
015D	E0	253	MOVX A,CDPTH	121	
015E	A3	254	INC UPTR	121	
015F	E0	255	MOVX A,CDPTH	121	
0160	A3	256	INC UPTR	121	
0161	E0	257	MOVX A,CDPTH	121	
0162	A3	258	INC UPTR	121	
0163	E0	259	MOVX A,CDPTH	121	
0164	A3	260	INC UPTR	121	
0165	E0	261	FIFTY: MOVX A,CDPTH	121	
0166	A3	262	INC UPTR	121	

AP-223

MCS-51 MACRO ASSEMBLER

CRTASM

MATHS

MATHS-2 MACRO ASSEMBLER

LUC	UPJ	LINE	SOURCE	SOURCE	LINE	LUC	UPJ
0167	E0	263	MOVX A, A, DPTM	MOVX	263	0167	E0
0168	A3	264	INC UPTR	INC	264	0168	A3
0169	E0	265	MOVX A, A, DPTM	MOVX	265	0169	E0
016A	A3	266	INC UPTR	INC	266	016A	A3
016B	E0	267	MOVX A, A, DPTM	MOVX	267	016B	E0
016C	A3	268	INC UPTR	INC	268	016C	A3
016D	E0	269	MOVX A, A, DPTM	MOVX	269	016D	E0
016E	A3	270	INC UPTR	INC	270	016E	A3
016F	E0	271	MOVX A, A, DPTM	MOVX	271	016F	E0
0170	A3	272	INC UPTR	INC	272	0170	A3
0171	E0	273	MOVX A, A, DPTM	MOVX	273	0171	E0
0172	A3	274	INC UPTR	INC	274	0172	A3
0173	E0	275	MOVX A, A, DPTM	MOVX	275	0173	E0
0174	A3	276	INC UPTR	INC	276	0174	A3
0175	E0	277	MOVX A, A, DPTM	MOVX	277	0175	E0
0176	A3	278	INC UPTR	INC	278	0176	A3
0177	E0	279	MOVX A, A, DPTM	MOVX	279	0177	E0
0178	A3	280	INC UPTR	INC	280	0178	A3
0179	E0	281	SIXTY: MOVX A, A, DPTM	MOVX	281	0179	E0
017A	A3	282	INC UPTR	INC	282	017A	A3
017B	E0	283	MOVX A, A, DPTM	MOVX	283	017B	E0
017C	A3	284	INC UPTR	INC	284	017C	A3
017D	E0	285	MOVX A, A, DPTM	MOVX	285	017D	E0
017E	A3	286	INC UPTR	INC	286	017E	A3
017F	E0	287	MOVX A, A, DPTM	MOVX	287	017F	E0
0180	A3	288	INC UPTR	INC	288	0180	A3
0181	E0	289	MOVX A, A, DPTM	MOVX	289	0181	E0
0182	A3	290	INC UPTR	INC	290	0182	A3
0183	E0	291	MOVX A, A, DPTM	MOVX	291	0183	E0
0184	A3	292	INC UPTR	INC	292	0184	A3
0185	E0	293	MOVX A, A, DPTM	MOVX	293	0185	E0
0186	A3	294	INC UPTR	INC	294	0186	A3
0187	E0	295	MOVX A, A, DPTM	MOVX	295	0187	E0
0188	A3	296	INC UPTR	INC	296	0188	A3
0189	E0	297	MOVX A, A, DPTM	MOVX	297	0189	E0
018A	A3	298	INC UPTR	INC	298	018A	A3
018B	E0	299	MOVX A, A, DPTM	MOVX	299	018B	E0
018C	A3	300	INC UPTR	INC	300	018C	A3
018D	E0	301	SEVENTY: MOVX A, A, DPTM	MOVX	301	018D	E0
018E	A3	302	INC UPTR	INC	302	018E	A3
018F	E0	303	MOVX A, A, DPTM	MOVX	303	018F	E0
0190	A3	304	INC UPTR	INC	304	0190	A3
0191	E0	305	MOVX A, A, DPTM	MOVX	305	0191	E0
0192	A3	306	INC UPTR	INC	306	0192	A3
0193	E0	307	MOVX A, A, DPTM	MOVX	307	0193	E0
0194	A3	308	INC UPTR	INC	308	0194	A3
0195	E0	309	MOVX A, A, DPTM	MOVX	309	0195	E0
0196	A3	310	INC UPTR	INC	310	0196	A3
0197	E0	311	MOVX A, A, DPTM	MOVX	311	0197	E0
0198	A3	312	INC UPTR	INC	312	0198	A3
0199	E0	313	MOVX A, A, DPTM	MOVX	313	0199	E0
019A	A3	314	INC UPTR	INC	314	019A	A3
019B	E0	315	MOVX A, A, DPTM	MOVX	315	019B	E0
019C	A3	316	INC UPTR	INC	316	019C	A3
019D	E0	317	MOVX A, A, DPTM	MOVX	317	019D	E0

CXTASM.

LUC	OBJ	LINE	SOURCE	ADDRESS	OBJ
019E	A3	318	INC A	019E	019E
019F	E0	319	MOVX A,CDPTN	019F	019F
01A0	A3	320	INC A	01A0	01A0
01A1	E0	321	EIGHTY: MOVX A,CDPTN	01A1	01A1
01A2	A3	322	INC A	01A2	01A2
		323			
01A3	E583	324	CHECK: MOV A,DPH	01A3	01A3
01A5	B41FVC	325	CJNE A,#1FH,DONE	01A5	01A5
01A6	E582	326	MOV A,DPL	01A6	01A6
01AA	B40047	327	CJNE A,#000H,DONE	01AA	01AA
01AU	750018	F 328	MOV MASTER,#18h	01AU	01AU
01BU	750000	F 329	MOV MASTER+1,#00H	01BU	01BU
01B3	22	330	RET	01B3	01B3
		331			
01B4	B58340	F 332	DONE: MOV MASTER,DPH	01B4	01B4
01B7	B58240	F 333	MOV MASTER+1,DPL	01B7	01B7
01BA	22	334	KEI	01BA	01BA
		335			
01BB	C3	336	DMADNE: CLR C	01BB	01BB
01BC	E582	337	MOV A,DPL	01BC	01BC
01BE	244F	338	ADD A,#79D	01BE	01BE
01CU	F582	339	DPL,A	01CU	01CU
01C2	B0DF	340	JNC CHECK	01C2	01C2
01C4	B583	341	INC DPH	01C4	01C4
01C6	B0DB	342	SJMP CHECK	01C6	01C6
		343			
		344			
		345	END		

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
ACC. . . .	D ADDR	00E0H	A
BLANK. . .	C ADDR	0085H	A PUB
BLINE. . .	C ADDR	00A8H	A PUB
BUFFER. . .	C ADDR	003FH	A
CHECK. . .	C ADDR	01A3H	A
CUNT1. . .	C ADDR	00B0H	A
CUNT2. . .	C ADDR	00E7H	A
CUUNT. . .	D ADDR	----	EXT
CURSER. . .	D ADDR	----	EXT
DONE. . . .	C ADDR	00F8H	A
DMA.	C ADDR	00FAH	A
DMADNE. . .	C ADDR	0180H	A
DONE. . . .	C ADDR	0184H	A
DPH.	D ADDR	0083H	A
DPL.	D ADDR	0082H	A
EIGHTY. . .	C ADDR	01A1H	A
ESCSEQ. . .	B ADDR	----	EXT
FIFO.	D ADDR	----	EXT
FIFTY. . . .	C ADDR	0165H	A
FILL.	C ADDR	00D0H	A PUB
FURTY. . . .	C ADDR	0151H	A
GUBACK. . .	C ADDR	0084H	A
L.	D ADDR	----	EXT
LINE0. . . .	D ADDR	----	EXT
NUTYET. . .	C ADDR	0097H	A
OVER.	C ADDR	0059H	A
OVER1. . . .	C ADDR	0078H	A
PINT.	D ADDR	----	EXT
PSW.	D ADDR	00D0H	A
RASIER. . .	D ADDR	----	EXT
SBUF.	D ADDR	0099H	A
SCAN.	B ADDR	----	EXT
SERBUF. . .	C ADDR	0052H	A
SERIAL. . .	D ADDR	----	EXT
SERINT. . .	B ADDR	----	EXT
SEVENTY. . .	C ADDR	0180H	A
SIXTY. . . .	C ADDR	0179H	A
TEN.	C ADDR	0115H	A
THIRTY. . .	C ADDR	0130H	A
TRNINI. . .	B ADDR	----	EXT
TWENTY. . .	C ADDR	0129H	A
VERI.	C ADDR	0025H	A

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

IS10-II MCS-51 MACRO ASSEMBLER v2.1
 OBJECT MODULE PLACED IN :F1:KEYBD.OBJ
 ASSEMBLER INVOKED BY: ASP51 :F1:KEYBD.SRC

SYMBOL TABLE LISTING

LOC	OBJ	LINE	SOURCE	VALUE	TYPE	NAME
1				A	HEX	0000
2				0000	HEX	A
3				0000	HEX	A
4			*****			
5			*****			
6			*****			
7			*****			
8			*****			
9			*****			
10			*****			
11			*****			
12			*****			
13			*****			
14			*****			
15			*****			
16			*****			
17			*****			
18			*****			
19			*****			
20			*****			
21			*****			
22			*****			
23			*****			
24			*****			
25			*****			
26			*****			
27			*****			
28			*****			
29			*****			
30			*****			
31			*****			
32			*****			
33			*****			
34			*****			
35			*****			
36	+1		SEJECT			

MCS-51 MACRO ASSEMBLER KEYBD

00137 00000000 0000 12-2000

LUC OBJ LINE SOURCE

00000000 0000 12-2000

```

3/
38 UNDECODED_KEYBOARD_SEGMENT CODE
---- 39 KSEG UNDECODED_KEYBOARD
40
41
42
43 HEADER: PUSH PSW ;PUSH REG USED BY PLM51
0000 C000 44 PUSH ACC
0002 C0E0 45 PUSH DPL
0004 C082 46 PUSH DPH
0006 C083 47 PUSH U0H
0008 C000 48
000A C001 49
000C C002 50
000E C003 51
0010 9010FF 52
53
54 MOV R1,#00H ;CLR ZERO COUNTER
0013 7900 55 MOV R0,#LSIKEY ;GET KEYBOARD RAM POINTER
0015 7800 F 56 MOV R3,#08H ;INITIALIZE LOOP COUNTER
0017 7B08 57 CLR KEY0 ;INITIALIZE PLM51 STATUS BITS
0019 C200 F 58 SETB SAME
001B 0200 F 59
001D 8602 60 MORE: MOV 02H,R0 ;MOV LAST KEYBOARD SCAN TO 02H
001F E4 61 CLR A
0020 93 62 MOV A,A+DPTH ;SCAN KEYBOARD
0021 F4 63 CPL A ;INVERT
0022 6005 64 JZ ZERO ;IF SCAN WAS ZERO GO INCREMENT ZERO COUNTER
0024 B50224 65 CJNE A,02H,NTSAME ;COMPARE WITH LAST SCAN IF NOT THE SAME
66 ;THEN CLR SAME BIT AND WRITE NEW INFORMATION
67 ;TO RAM
68 ;IF EQUAL JMP OVER INCR OF ZERO COUNTER
0027 8005 69 SJMP EQUAL ;INCH ZERO COUNTER
0029 0501 70 ZERO: INC 01H
002B B50210 71 CJNE A,02H,NTSAME
002E 08 72 EQUAL: INC R0 ;STEP TO NEXT SCAN RAM LOCATION
002F 0583 73 INC DPH ;NEXT KEYBOARD ADDRESS
0031 0REA 74 DJNZ R3,MORE ;IF LOOP COUNTER NOT 0, SCAN AGAIN
0033 B90804 75 CJNE R1,#08H,BACK ;CHECK TO SEE IF ALL 8 SCANS WHERE 0
0036 0200 F 76 SETB KEY0 ;IF YES SET KEY0 BIT
0038 C200 F 77 CLR SAME
003A 0003 78 BACK: POP 03H
003C 0002 79 POP 02H ;POP REGISTERS
003E 0001 80 POP 01H
0040 0000 81 POP 00H
0042 0083 82 POP DPH
0044 0082 83 POP DPL
0046 00E0 84 POP ACC
0048 0000 85 POP PSW
004A 22 86 KET
87
88
89
90
91 END
004B F6 85 NTSAME: MOV R0,A ;IF SCAN WAS NOT THE SAME THEN PUT NEW
86 ;SCAN INFO INTO RAM
004C C200 F 87 CLR SAME ;CLR SAME BIT
004E 800E 88 SJMP EQUAL ;GO DO MORE
89
90
91

```

KEYWORD

NAME	TYPE	VALUE	ADDRESS
ACC	D	00E0H	A
BACK	C	003AH	R
DPH	D	0083H	A
DPL	D	0082H	A
EQUAL	C	002EH	R
KEYU	E	----	EXT
LSTKEY	D	----	EXT
MORE	C	0010H	R
NISAME	C	0040H	R
PSW	D	0000H	A
READER	C	0000H	R
SAME	E	----	EXT
UNDECODED_KEYBOARD	C	0050H	R
ZERU	C	0029H	R
REGISTER BANK(S) USED: 0			
ASSEMBLY COMPLETE, NO ERRORS FOUND			

MCS-51 MACRO ASSEMBLER DECODE

ISIS-1I MCS-51 MACRO ASSEMBLER V2.1
 OBJECT MODULE PLACED IN :F1:DECODE.OBJ
 ASSEMBLER INVOKED BY: ASM51 :F1:DECODE.SRC

LUC	OBJ	LINE	SOURCE
		1	
		2	
		3	
		4	*****
		5	*****
		6	*****
		7	SOFTWARE FOR DECODED KEYBOARD
		8	*****
		9	*****
		10	*****
		11	*****
		12	*****
		13	*****
		14	*****
		15	PUBLIC DETACH
		16	EXTRN DATA (LSTKEY)
		17	EXTRN BIT (KBDINT)
		18	
		19	
		20	*****
		21	*****
		22	*****
		23	*****
		24	*****
		25	*****
		26	+1 SEJECT

```

LCC OBJ      LINE  SOURCE
-----
27
28 DECODED_KEYBOARD SEGMENT CODE
29 MSEG DECODED_KEYBOARD
30
0000 C000    31 DETACH: PUSH    PSW          ;PUSH REGISTERS
0002 C002    32          PUSH    DPL          ;USED BY PLM51
0004 C004    33          PUSH    DPH
0006 C006    34          PUSH    ACC
0008 4080FF  35          MCV     DP1R,#80FFH    ;ADDRESS FOR KEYBOARD
000A E4      36          CLK     A
000C 93      37          MOVC   A,@A+DPTH    ;FETCH ASCII BYTE
000E F500    38          MOV     DSIKEY+1,A    ;MOV TO MEMORY TO BE READ BY PLM51
0010 0200    39          SETB   KBINT      ;LET PLM51 KNOW THERE IS A BYTE
0012 750CFF  40          MOV     TP0,#0FFH    ;SET COUNTER TO FFFFH SO INTERRUPT
0014 750AFF  41          MOV     TL0,#0FFH    ;ON THE NEXT FALLING EDGE OF T0
0016 00E0    42          POP     ACC
0018 0003    43          POP     DPH          ;POP REGISTERS
001A 0002    44          POP     DPL
001C 0000    45          POP     PSW
001E 32      46          RETI
47
48
49
50
51 END

```


MCS-51 MACRO ASSEMBLER DECODE

MCS-51 MACRO ASSEMBLER DECODE

SYMBOL TABLE LISTING

1.54 REVISION 1.54
 1.54 REVISION 1.54
 1.54 REVISION 1.54
 1.54 REVISION 1.54

NAME	TYPE	VALUE	ATTRIBUTES
ACC.	D ALCH	00E0H A	
DECODED_KEYBOARD	C SEG	0020H	REL=LNII
DETACH	C ALCH	0000H R PUB	SEG=DECODED_KEYBOARD
DPH.	D ALCH	0083H A	
DPL.	D ALCH	0082H A	
KBDINT.	B ALCH	----	EXT
LSTKEY	D ALCH	----	EXT
PSW.	D ALCH	00D0H A	
TH0.	D ALCH	008CH A	
TL0.	D ALCH	008AH A	

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

THIS CONTAINS THE SOFTWARE NEEDED TO PERFORM A SERIAL SEARCH
 FOR THE SERIAL NUMBER AND DETACHABLE KEYBOARD. THIS PROGRAM MUST
 BE LINKED TO THE MAIN PROGRAM FOR USE.

MCS-51 MACRO ASSEMBLER DETACH

ISIS-II MCS-51 MACRO ASSEMBLER V2.1
 OBJECT MODULE PLACED IN :F1:DETACH.OBJ
 ASSEMBLER INVOKED BY: ASM51 :F1:UEIACH.SRC

```

LUC OBJ LINE SOURCE
1
2
3
4 ;*****
5 ;*****
6 ;****
7 ;**** SOFTWARE FOR A SERIAL OR DETACHABLE ****
8 ;**** KEYBOARD ****
9 ;****
10 ;*****
11 ;*****
12 ;
13 ;
14 ;
15 ; THIS CONTAINS THE SOFTWARE NEEDED TO PERFORM A SOFTWARE SERIAL
16 ; PORT FOR SERIAL KEYBOARDS AND DETACHABLE KEYBOARD. THIS PROGRAM MUST
17 ; BE LINKED TO THE MAIN PROGRAMS FOR USE.
18 ;
19 +1 SEJECT

```

MCS-51 MACRO ASSEMBLER DETACH

LUC	OBJ	LINE	SOURCE
		20	;
		21	;
00B4		22	INPUT EQU TO
		23	
		24	
		25	
		26	PUBLIC DETACH
		27	EXTERN DATA (LSIKEY)
		28	EXTERN BIT (RCVFLG, SYNC, BYFIN)
		29	EXTERN BIT (KBDINI, ERKOK)
		30	
		31	;
		32	;
		33	;
		34	
		35	;
		36	;
		37	;
		38	;
		39	;
		40	BAUD START BIT DETECT MESSAGE DETECT
		41	110 0EFA2H 0DF45H
		42	150 0F400H 0E800H
		43	;
		44	;
		45	;
		46	;
0000		47	START0 EQU 000H ;LOW BYTE FOR 150 BAUD
00F4		48	START1 EQU 0F4H ;HIGH BYTE FOR 150 BAUD
0000		49	MESSAGE0 EQU 000H ;LOW BYTE FOR 150 BAUD
00E8		50	MESSAGE1 EQU 0E8H ;HIGH BYTE FOR 150 BAUD
		51	+1 SEJECT

MCS-51 MACRO ASSEMBLER DETACH

```

LCC OBJ      LINE      SOURCE
52
53
54 ;*****
55 ;*
56 ;*          "DETACH" INTERRUPT ROUTINE FOR DETACHABLE KEYBOARDS
57 ;*
58 ;*****
59
60
61
62 DETACHABLE_KEYBOARD SEGMENT CODE
63 KSEG DETACHABLE_KEYBOARD
64
65 DETACH: PUSH    PSW          ;PUSH REGISTERS USED BY PLM51
66          PUSH    ACC
67          JB      RCvFLG,VALID ;IF RECEIVE FLAG SET GET NEXT BIT
68          JB      INPL1,MS1    ;IF TO IS A 1 THEN NOT A START BIT
69          SETB    RCvFLG      ;IF TO IS 0 THEN IT A START BIT
70          MOV     TH0,#SIANT1  ;SET TIMER TO INTERRUPT IN THE MIDDLE OF START BIT
71          MOV     TLO,#SIANTU
72          MOV     A,TH0C
73          CLM     UE2H        ;SET TIMER COUNTER TO TIMER MODE
74          MOV     TH0D,A
75          SJMP    FINI        ;GO BACK TO PRUGNAM
76
77 ;
78 VALID:   JB      SYNC,NXTBIT ;CHECK IF VALID START BIT HAS BEEN SEEN
79          JB      INPL1,MS1    ;IF NOT CHECK IF VALID START BIT
80          SETB    SYNC        ;IF YES SET SYNC
81          MOV     LSTKEY,#00H  ;INIT LSTKEY
82          MOV     TH0,#MESSAGE1
83          MOV     TLO,#MESSAGE0 ;SET TIMER FOR 1 BIT TIME
84          SJMP    FINI        ;AND GO BACK TO MAIN PROGRAM
85
86 ;
87 NXTBIT:  MOV     TH0,#MESSAGE1 ;SET TIMER FOR 1 BIT TIME
88          MOV     TLO,#MESSAGE0 ;CHECK TO SEE IF ALL 8 BITS HAVE BEEN RECEIVED
89          JB      BYFIN,STUP    ;GET WORKING REGISTER
90          MOV     A,LSTKEY      ;GET NEXT BIT FROM T1
91          RRC     A
92          MOV     LSIKEY,A
93          JNC     FINI          ;IF NO CARRY THEN NOT DONE
94          SETB    BYFIN
95          CLM     UE7H        ;CLR BIT 7
96          MOV     LSIKEY+1,A    ;MOV FINAL CODE TO LSTKY+1
97          FINI:   POP     ACC
98          POP     PSW
99          RETI

```


MCS-51 MACRO ASSEMBLER DETACH

LUC	OBJ	LINE	SOURCE
		99	;
004A	30b405	100	STUP: JNB INPL,ERN
004D	0200	101	SETB ABUIN
004F	020000	102	JMP KSI
		103	;
0052	0200	104	ERN: SETB ERNCK
0054	0200	105	KSI: CLR KCVFLG
0056	0200	106	CLR SYNC
0058	0200	107	CLR BYPIN
005A	E509	108	MOV A,INCD
005C	0202	109	SETB UE2
005E	F509	110	MOV INC,A
0060	750CF	111	MOV IPU,NOFFH
0063	750AFF	112	MOV ILU,NOFFH
0066	800D	113	SJMP FINI
		114	
		115	
		116	
		117	
		118	END

;IF NOT 1 THEN NOT A VALID STOP BIT
;TELL PLM A BYTE IS READY
;AND GO BACK TO MAIN PROGRAM

;CLEAR FLAGS

;SET TIMER 0 TO COUNTER MODE

;SET COUNTER TO FFFFH SO INTERRUPT
;ON NEXT FALLING EDGE OF T0

MCS-51 MACRO ASSEMBLER DETACH

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
ACC	C ADDR	00E0H	A
BYFIN	B ADDR	----	EXT
DETACH	C ADDR	0000H	R PUB
DETACHABLE_KEYBOARD	C SEG	0068H	SEG=DETACHABLE_KEYBOARD REL=UNIT
ERR	C ADDR	0052H	R
ERRUR	B ADDR	----	EXT
FINI	C ADDR	0045H	R
INPUT	B ADDR	00B0H.4	A
KBDINT	B ADDR	----	EXT
LSTKEY	B ADDR	----	EXT
MESSAGEV	NLMB	0000H	A
MESSAGE1	NLMB	00E8H	A
NXTBIT	C ADDR	002DH	R
PSW	B ADDR	00D0H	A
RCVFLG	B ADDR	----	EXT
RST	C ADDR	0054H	R
STAKT0	NLMB	0000H	A
STAKT1	NLMB	00F4H	A
STOP	C ADDR	004AH	R
SYNC	B ADDR	----	EXT
TU	B ADDR	00B0H.4	A
TH0	B ADDR	008CH	A
TLO	B ADDR	008AH	A
TMOU	B ADDR	0089H	A
VALID	C ADDR	001AH	R

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

APPENDIX B REFERENCES

1. John Murray and George Alexy, *CRT Terminal Design Using The Intel 8275 and 8279*, Intel Application Note AP-32, Nov., 1977.
2. John Katausky, *A Low Cost CRT Terminal Using The 8275*, Intel Application Note AP-62, Nov., 1979..

Interfacing the 82786 Graphics
Coprocessor to the 8051

RICK SCHUE
REGIONAL APPLICATIONS SPECIALIST
INDIANAPOLIS, INDIANA

1. John Murray and George Alexy, "CRT Terminal Design Using the Intel 8275 and 8255", Intel Application Note AP-53, Nov. 1977.
2. John Katsenky, "A Low Cost CRT Terminal Using the 8275", Intel Application Note AP-61, Nov. 1979.

August 1990

Interfacing the 82786 Graphics Coprocessor to the 8051

RICK SCHUE
REGIONAL APPLICATIONS SPECIALIST
INDIANAPOLIS, INDIANA

Order Number: 270528-003

INTERFACING THE 82786 GRAPHICS COPROCESSOR TO THE 8051

For write cycles, the 8051 writes the low byte of the word into the latch at address C000H and the high byte into address C001H. Next the upper address bits are set with PORT1 and a dummy write cycle is performed in graphics memory at the desired address (A000H-BFFFH). Like in the read example, the 82786 runs a memory cycle at this point, enabling the output of the latching transceiver at the proper time in the write cycle, as indicated by SEN going active.

Accessing the registers inside the 82786 is done in exactly the same fashion, except that the 82786 is addressed in locations 8000H through 807EH. This causes the EPLD to drive the M/IO pin low during these cycles.

DESIGN NOTES

MP2433 are used for the latching transceivers in this design, although JAHCT046's would be used to reduce the total power consumption. Some changes to the EPLD would be required in this case. The interface assumes that all memory accesses to the 82786 are word references; accordingly, BHE is grounded at the 82786. All addresses generated by the 8051 must be even byte addresses; the only byte operations allowed are the reads and writes to the latching transceivers. The design shown here incorporates hardware work-arounds for the earlier "C-step," 82786; the current "D-step" part will work in the design as well. Additional information regarding the 82786 can be found in the "82786 Graphics Coprocessor User's Manual," Intel publication number 33193.

CONTENTS	PAGE
HARDWARE	2-156
OPERATION	2-156
DESIGN NOTES	2-156

also required in the external logic to allow the 8051 to read and write the 16 bit graphics memory supported by the 82786. This byte swapping is accomplished with the latching transceiver as well. All of the control logic is implemented in an Intel 70060 EPLD, allowing the entire interface to fit into three 24 pin DIPs.

HARDWARE

2

Figure 1 shows the interface between the 8051 bus and the 82786. Figure 2 shows a typical 8051 CPU design needed to complete the circuit. In this design the 82786 is mapped into an 8K byte window in 8051 data memory space. The upper address bits are used as a "page select," and are provided by I/O pins on the 8051. The 8051 EPLD contains the control logic for the transceivers and address decoding for the 82786. An address latch circuit for the EPLD is shown in Figure 3; the "ADR" signal is shown in Figure 4. The 82786 data memory is mapped into one 8K block (A000H-BFFFH); the 82786 registers are mapped into another (8000H-807EH) and the transceivers are mapped into a third block of memory (C000H-C001H).

OPERATION

Operation of the interface is as follows. For reading the graphics memory, the 8051 sets the upper address bits (PORT 1.0-1.3) and then performs a dummy read operation to the desired location in graphics memory (A000H from BFFFH). The dummy read cycle pro-

Interfacing the 82786 to the 8051 presents some interesting challenges, but can be accomplished with a little additional logic and software. Since the 82786 looks like a DRAM controller to the host CPU, wait states are often required when accessing the coprocessor. Since wait states are not supported by the 8051, latching transceivers and dummy read and write cycles are used to communicate with the 82786. Byte swapping is also required in the external logic to allow the 8 bit 8051 to read and write the 16 bit graphics memory supported by the 82786. This byte swapping is accomplished with the latching transceivers as well. All of the control logic is implemented in an Intel 5C060 EPLD, allowing the entire interface to fit into three 24 pin DIPs.

HARDWARE

Figure 1 shows the interface between the 8051 bus and the 82786. Figure 2 shows a typical 8051 CPU design needed to complete the circuit. In this design the 82786 is mapped into an 8K byte window in 8051 data memory space. The upper address bits are used as a "page select" and are provided by I/O pins on the 8051. The 5C060 EPLD contains the control logic for the transceivers and address decoding for the 82786. An equivalent circuit for the EPLD is shown in Figure 3; the ".ADF" file is shown in Figure 4. The 82786 data memory is mapped into one 8K block (A000H-BFFEh), the 82786 registers are mapped into another (8000H-807EH), and the transceivers are mapped into a third block of memory (C000H-C001H).

OPERATION

Operation of the interface is as follows. For reading the graphics memory, the 8051 sets the upper address bits (PORT 1.0-1.3) and then performs a dummy read operation to the desired location in graphics memory (A000H thru BFFEh). The dummy read cycle pro-

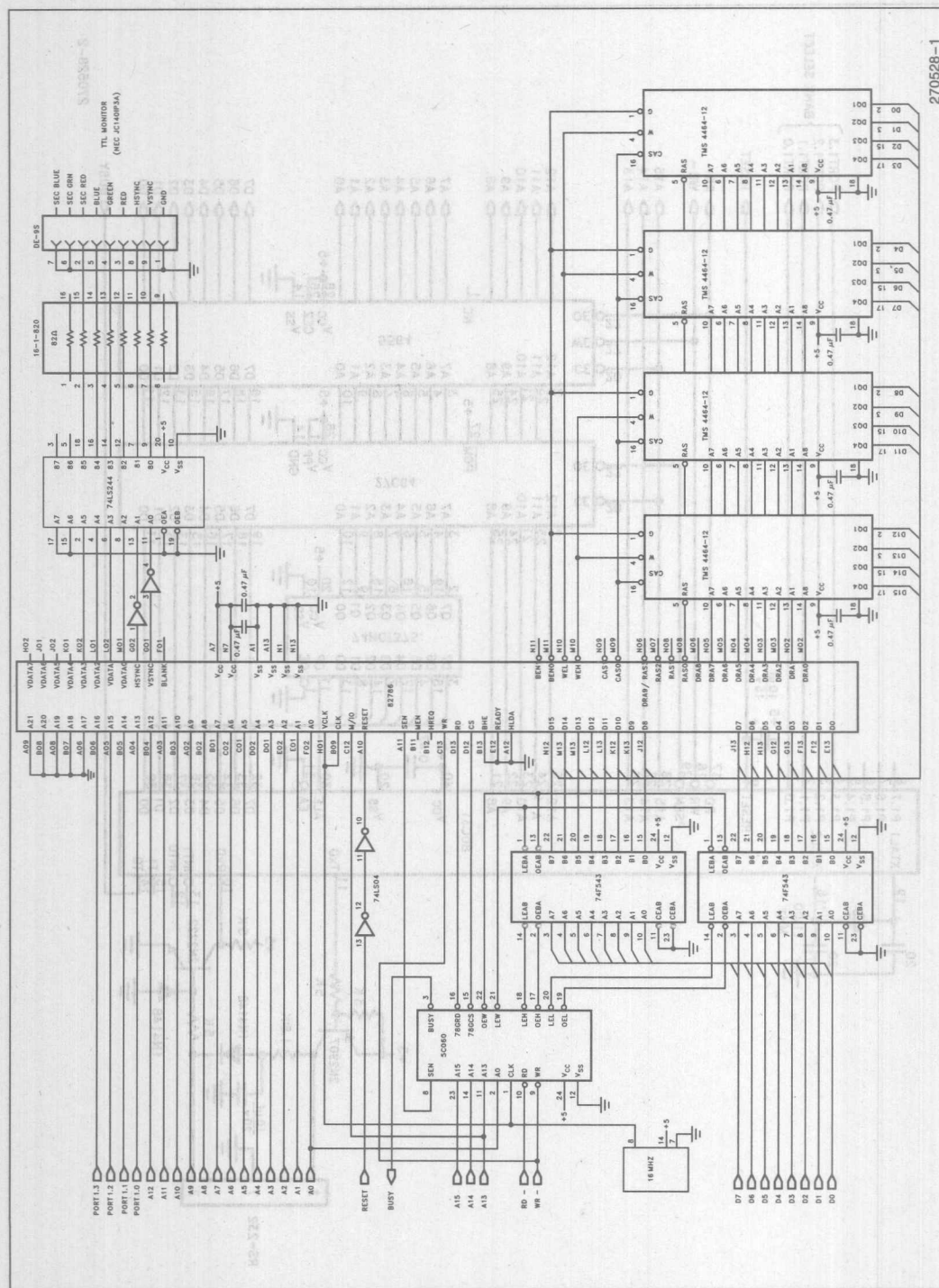
vides the address and RD/WR information to the 82786, which runs a cycle and deposits the 16 bit result into the latching transceivers at the end of the read cycle, as indicated by SEN. This event clears the BUSY flip flop in the EPLD. When the BUSY signal goes inactive, the 8051 reads the low byte from the latching transceiver at address C000H and the high byte free address C001H.

For write cycles, the 8051 writes the low byte of the word into the latch at address C000H and the high byte into address C001H. Next the upper address bits are set with PORT1 and a dummy write cycle is performed in graphics memory at the desired address (A000H-BFFEh). Like in the read example, the 82786 runs a memory cycle at this point, enabling the outputs of the latching transceivers at the proper time in the write cycle, as indicated by SEN going active.

Accessing the registers inside the 82786 is done in exactly the same fashion, except that the 82786 is addressed in locations 8000H through 807EH. This causes the EPLD to drive the M/I/O pin low during these cycles.

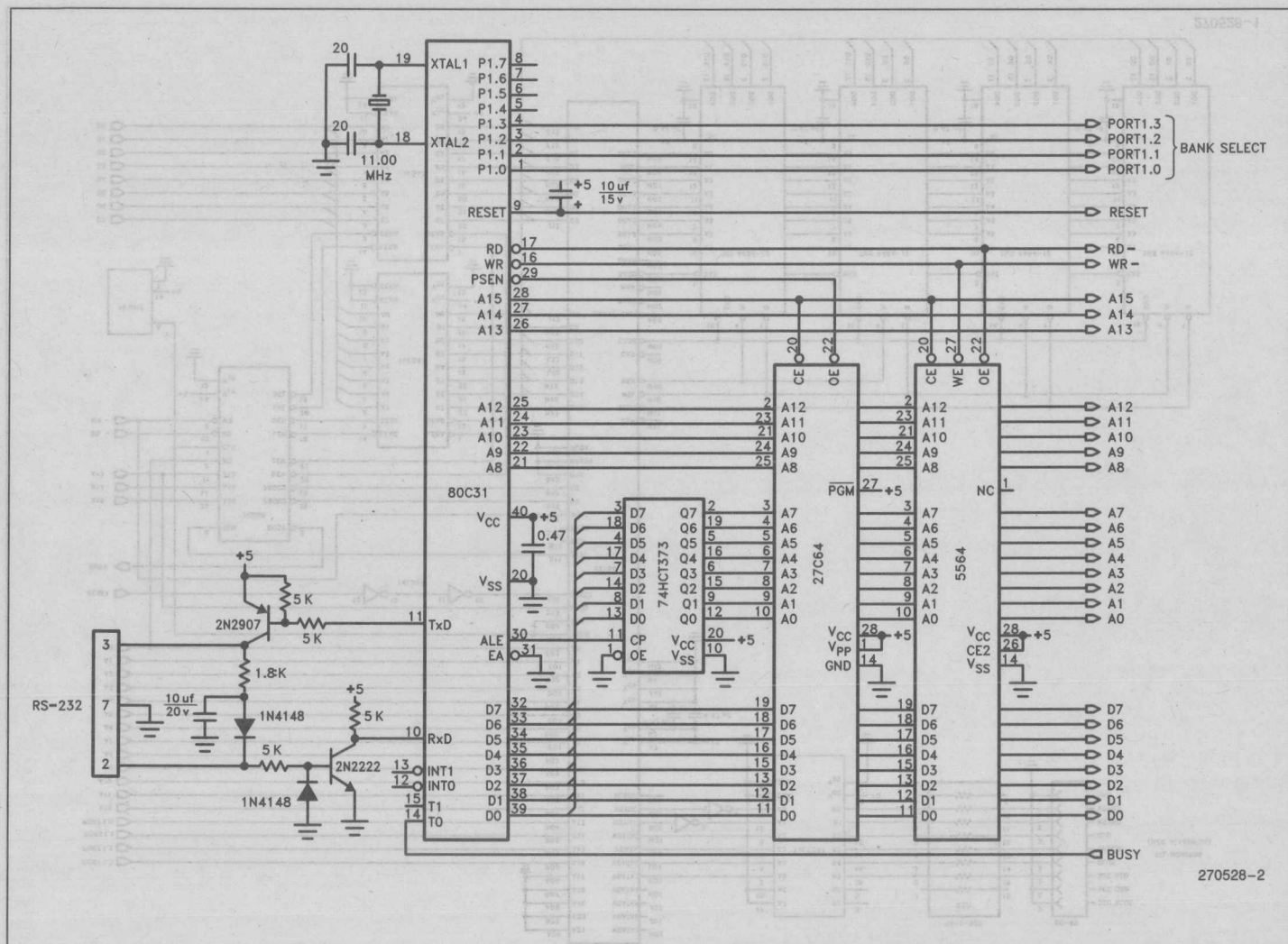
DESIGN NOTES

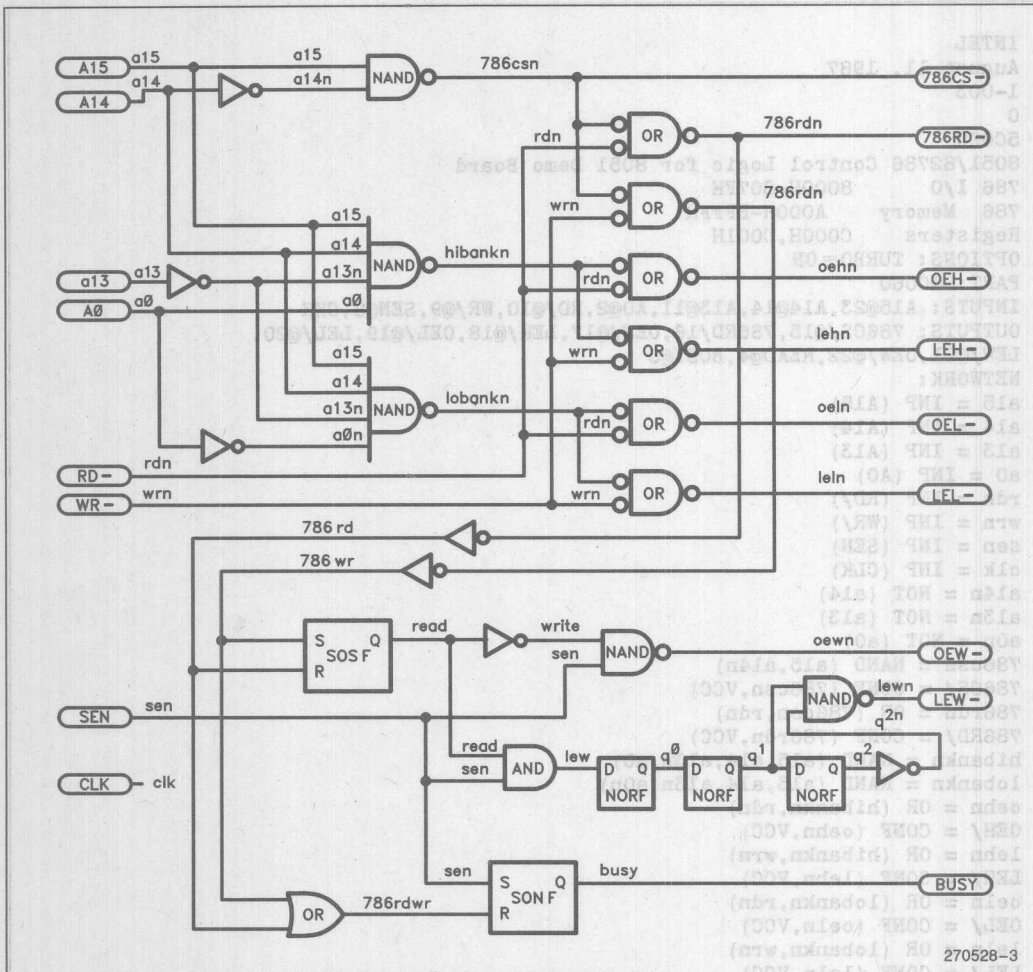
74F543's are used for the latching transceivers in this design, although 74HCT646's could be used to reduce the total power consumption. Some changes to the EPLD would be required in this case. The interface assumes that all memory accesses to the 82786 are word references; accordingly, BHE is grounded at the 82786. All addresses generated by the 8051 must be even byte addresses, the only byte operations allowed are the reads and writes to the latching transceivers. The design shown here incorporates hardware work-arounds for the earlier "C-step" 82786; the current "D-step" part will work in the design as well. Additional information regarding the 82786 can be found in the "82786 Graphics Coprocessor User's Manual", Intel publication number 231933.



2

Figure 1
2-157





2

Figure 3. 8051/82786 Control EPLD (5C060-55) Equivalent Circuit

INTL

August 11, 1987

1-003

0

5C060

8051/82786 Control Logic for 8051 Demo Board

786 I/O 8000H-807FH

786 Memory A000H-BFFFH

Registers C000H,C001H

OPTIONS: TURBO=ON

PART: 5C060

INPUTS: A15@23,A14@14,A13@11,A0@2,RD/@10,WR/@9,SEN@8,CLK

OUTPUTS: 786CS/@15,786RD/16,OEH/@17,LEH/@18,OEL/@19,LEL/@20,

LEW/@21,OEW/@22,READ@4,BUSY@3

NETWORK:

a15 = INP (A15)

a14 = INP (A14)

a13 = INP (A13)

a0 = INP (A0)

rdn = INP (RD/)

wrn = INP (WR/)

sen = INP (SEN)

clk = INP (CLK)

a14n = NOT (a14)

a13n = NOT (a13)

a0n = NOT (a0)

786csn = NAND (a15,a14n)

786CS/ = CONF (786csn,VCC)

786rdn = OR (786csn,rdn)

786RD/ = CONF (786rdn,VCC)

hibankn = NAND (a15,a14,a13n,a0)

lobankn = NAND (a15,a14,a13n,a0n)

oehn = OR (hibankn,rdn)

OEH/ = CONF (oehn,VCC)

lehn = OR (hibankn,wrn)

LEH/ = CONF (lehn,VCC)

oeln = OR (lobankn,rdn)

OEL/ = CONF (oeln,VCC)

leln = OR (lobankn,wrn)

LEL/ = CONF (leln,VCC)

oewn = NAND (sen,write)

OEW/ = CONF (oewn,VCC)

lew = AND (sen,read)

q0 = NORF (lew,clk,GND,GND)

q1 = NORF (q0,clk,GND,GND)

q2 = NORF (q1,clk,GND,GND)

q2n = NOT (q2)

lewn = NAND (q1,q2n)

LEW/ = CONF (lewn,VCC)

786wr = NOR (786csn,wrn)

786rd = NOT (786rdn)

READ,read = SOSF (786rd,clk,786wr,GND,GND,VCC)

write = NOT (read)

786rdwr = OR (786rd,786wr)

BUSY = SONF (786rdwr,clk,sen,GND,GND,VCC)

END\$

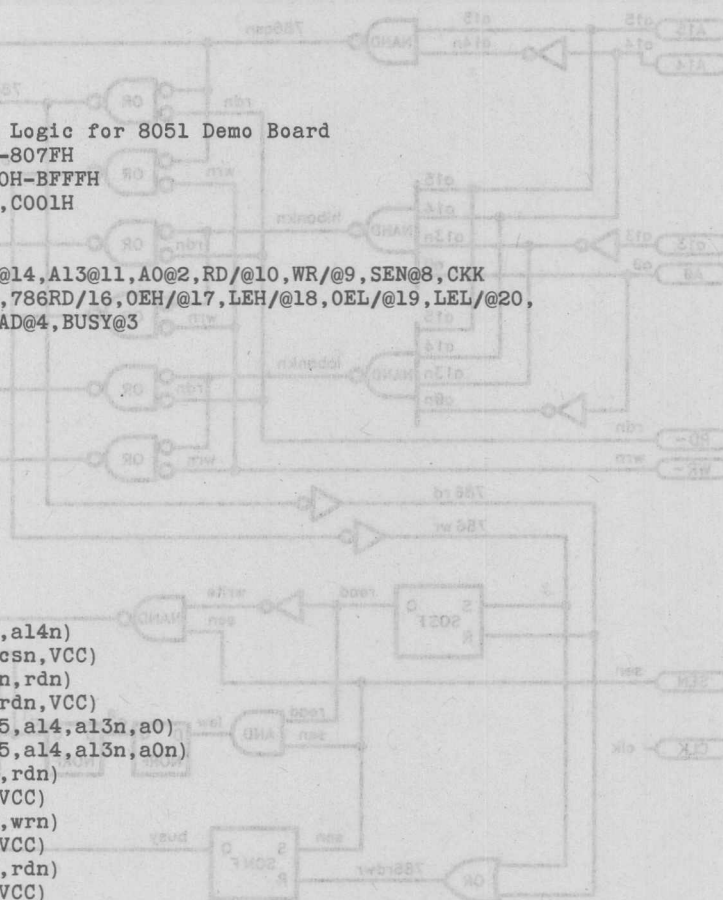


Figure 4.

INTRODUCTION	2-163
HARDWARE DESIGN	2-163
TIMING REQUIREMENTS	2-163
SOFTWARE	2-163
REFERENCES	2-163

8051 DENSITRON LCD TO THE

September 1990

2

Interfacing the Densitron LCD to the 8051

RICK SCHUE
REGIONAL APPLICATIONS SPECIALIST
INDIANAPOLIS, INDIANA

Order Number: 270529-003

**INTERFACING THE
DENSITRON LCD TO THE
8051**

CONTENTS

PAGE

INTRODUCTION	2-163
HARDWARE DESIGN	2-163
TIMING REQUIREMENTS	2-163
SOFTWARE	2-163
REFERENCES	2-163

September 1990

5

**Interfacing the Densitron LCD to
the 8051**

**RICK SCHUE
REGIONAL APPLICATIONS SPECIALIST
INDIANAPOLIS, INDIANA**

Order Number 270529-003

INTRODUCTION

This application note details the interface between an 80C31 and a Densitron two row by 24 character LM23A2C24CBW display. This combination provides a very flexible display format (2x24) and a cost effective, low power consumption microcontroller suitable for many industrial control and monitoring functions.

Although this applications brief concentrates on the 80C31, the same software and hardware techniques are equally valid on other members of the 8051 family, including the 8031, 8751, and the 8044.

HARDWARE DESIGN

The LCD is mapped into external data memory, and looks to the 80C31 just like ordinary RAM. The register select (RS) and the read/write (R/W) pins are connected to the low order address lines A0 and A1. Connecting the R/W pin to an address line is a little unorthodox, but since the R/W line has the same set-up time requirements as the RS line, treating the R/W pin as an address kept this pin from causing any timing problems.

The enable (E) pin of the LCD is used to select the device, and is driven by the logical OR of the 80C31's RD and WR signals AND'ed with the MSB of the address bus. This maps the LCD into the upper half of the 64 KB external data space. If this seems a little wasteful, feel free to use a more elaborate address decoding scheme.

With the address decoding shown in the example, the LCD is mapped as follows:

Address	Function	Read/Write?
8000H	Write Command to LCD	Write Only
8001H	Write Data to LCD	Write Only
8002H	Read Status from LCD	Read Only
8003H	Read Data from LCD	Read Only
8004H to FFFFH	No Access	

Undefined results may occur if the software attempts to read address 8000H or 8001H, or write to address 8002H or 8003H.

TIMING REQUIREMENTS

The timing requirements of the Densitron LCD are a little slow for a full speed 80C31. The critical timing parameters are the enable pulse width (PW E) of 450 ns, and the data delay time during read cycles (tDDR) of 320 ns. The 80C31 is available at clock speeds up to 16 MHz, but at this speed these parameters are violated. Since the 80C31 lacks a READY pin, the only way to satisfy the LCD timing requirements is to slow the clock down to 10 MHz or lower. A convenient crystal frequency is 7.3728 MHz since it allows all standard baud rates to be generated with the internal timers.

SOFTWARE

The code consists of a main module and a set of utility procedures that talk directly to the LCD. This way the application code does not have to be concerned with where the LCD is mapped, or the exact bit patterns needed to control it. The mainline consists of a call to initialize the LCD, and then it writes a message to the screen, waits, and then erases it. It repeats this indefinitely.

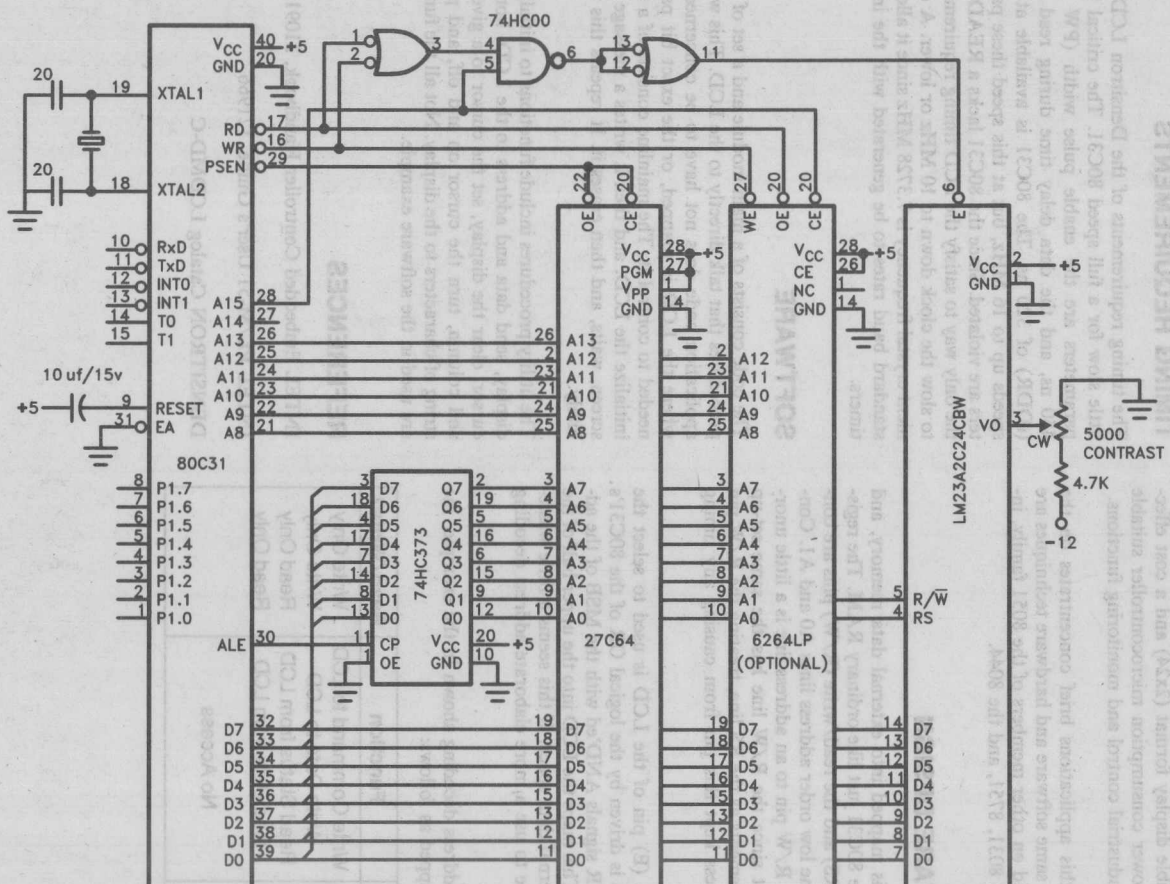
The utility procedures include functions to initialize the display, send data and address to the LCD, home the cursor, clear the display, set the cursor to a given row and column, turn the cursor on and off, and print a string of characters to the display. Not all the functions are used in the software example.

REFERENCES

INTEL Embedded Controller Handbook, 210918

INTEL PL/M-51 User's Guide, 121966

DENSITRON Catalog LCDMD-C



270529-1

Figure 1.
2-164

```
Main_module: DO;
```

```
Delay: PROCEDURE (count) EXTERNAL;  
  DECLARE count WORD;  
END Delay;
```

```
Initialize_LCD: PROCEDURE EXTERNAL;  
END Initialize_LCD;
```

```
Clear_display: PROCEDURE EXTERNAL;  
END Clear_display;
```

```
LCD_print: PROCEDURE EXTERNAL;  
END LCD_print;
```

```
DECLARE LCD_buffer (48) BYTE PUBLIC,  
  sign_on_message (*) BYTE CONSTANT  
  ('INTEL 8051 DRIVES LCD - '  
   '2 ROWS BY 24 CHARACTERS '),  
  1 BYTE;
```

```
/* This is the start of the program */
```

```
/* Initialize the LCD */  
CALL Initialize_LCD;  
CALL Clear_display;
```

```
/* Now enter an endless loop to display the message */  
DO WHILE 1;
```

```
/* Copy the message to the buffer */
```

```
DO i = 0 to 47;
```

```
  LCD_buffer(i) = sign_on_message(i);  
END;
```

```
/* Now print out the buffer to the LCD */  
CALL LCD_print;
```

```
/* wait a while */  
CALL Delay(2000);
```

```
/* now clear the screen */  
CALL Clear_display;
```

```
END; /* of DO WHILE */
```

```
END Main_module;
```

Main Module

2

```

LCD_IO_MODULE: DO;
MainModule: DO;

DECLARE LCD_buffer (48) BYTE EXTERNAL;
Delay: PROCEDURE (count) EXTERNAL;
LCD_command BYTE AT (08000H) AUXILIARY;
DECLARE LCD_data BYTE AT (08001H) AUXILIARY;
END Delay;
LCD_status BYTE AT (08002H) AUXILIARY;
Initialize_LCD: PROCEDURE;
LCD_busy LITERALLY '1000$0000B';
i BYTE;
END Initialize_LCD;

Delay: PROCEDURE (msec) PUBLIC;
Clear display: PROCEDURE EXTERNAL;
END Clear_display;

/* This procedure causes a delay of n msec */
LCD_print: PROCEDURE EXTERNAL;
END LCD_print;

DECLARE msec WORD,
i WORD;
PUBLIC BYTE (48)
DECLARE LCD_buffer
sign_on_message
CONSTANT BYTE (*)
/* INITEL BOOI DRIVES LCD -
'2 ROWS BY 24' 25 ROWS BY 24'
*/
IF msec > 0 THEN DO;
DO i = 0 to msec - 1;
CALL Time(5); /* .2 msec delay */
END;
END Delay;
/* This is the start of the program */

LCD_out: PROCEDURE (char) PUBLIC;
/* Initialize the LCD */
CALL Initialize_LCD;
CALL Clear_display;

DECLARE char BYTE;
/* Now enter an endless loop to display the message */
DO WHILE 1;
/* wait for LCD to indicate NOT busy */
/* Copy the message to the buffer */
DO i = 0 to 47;
LCD_buffer(i) = sign_on_message(i);
END;
/* now send the data to the LCD */
LCD_data = char;
/* Now print out the buffer to the LCD */
CALL LCD_print;
CALL Delay(2000);
/* wait a while */
/* now clear the screen */
CALL Clear_display;
END;
END MainModule;

```

LCD Driver Module


```

LCD_command_out: PROCEDURE (char) PUBLIC;

    DECLARE char BYTE;

    /* wait for LCD to indicate NOT busy */
    DO WHILE (LCD_status AND LCD_busy) <> 0;
    END;

    /* now send the command to the LCD */
    LCD_command = char;

END LCD_command_out;

Home_cursor: PROCEDURE PUBLIC;

    CALL LCD_command_out(0000$0010B);

END Home_cursor;

Clear_display: PROCEDURE PUBLIC;

    CALL LCD_command_out (0000$0001B);

END Clear_display;

Set_cursor: PROCEDURE (position) PUBLIC;

    DECLARE    position        BYTE;

    IF position > 47 THEN position = 47;
    IF position < 24 THEN CALL LCD_command_out(080H + position);
    ELSE CALL LCD_command_out(0C0H + (position - 24));

END Set_cursor;

Cursor_on: PROCEDURE PUBLIC;

    CALL LCD_command_out(0000$1111B);

END Cursor_on;

Cursor_off: PROCEDURE PUBLIC;

    CALL LCD_command_out(0000$1100B);

END Cursor_off;

```

LCD Driver Module (Continued)

```
LCD_print: PROCEDURE PUBLIC;
```

```
/* This procedure copies the contents of the LCD_buffer
to the display */
```

```
CALL Set_cursor(0) ;
DO i = 0 to 23;
    CALL LCD_out(LCD_buffer(i));
END;
CALL Set_cursor(24);
DO i = 24 to 47;
    CALL LCD_out(LCD_buffer(i));
END;
```

```
END LCD_print;
```

```
Initialize_LCD: PROCEDURE PUBLIC;
```

```
CALL Delay(100);
CALL LCD_command_out(38H); /* Function Set */
CALL LCD_command_out(38H);
CALL LCD_command_out(06H); /* entry mode set */
CALL Clear_display;
CALL Home_cursor;
CALL Cursor_off;
CALL Set_cursor(0);
```

```
END Initialize_LCD;
```

```
END LCD_IO_Module;
```

LCD Driver Module (Continued)

September 1990

2

32-Bit Math Routines for the 8051

RICK SCHUE
REGIONAL APPLICATIONS SPECIALIST
INDIANAPOLIS, INDIANA

Order Number: 270530-002

32-BIT MATH ROUTINES FOR THE 8051

CONTENTS

PAGE

APPLICATION 2-171

CODE SOURCE LISTINGS 2-172

September 1990

2

32-Bit Math Routines for the 8051

RICK SCHUE
REGIONAL APPLICATIONS SPECIALIST
INDIANAPOLIS, INDIANA

Order Number SY0830-003

Here are some easy to use 16- and 32-bit math routines that take the pain out of calculations such as PID loops, A/D calibration, linearization calculations and anything else that requires 32-bit accuracy.

The package is written to interface with PL/M-51. Parameters are passed as 16-bit words to the routines, which perform operations on a 32-bit "accumulator" resident in memory. The following functions are performed:

Load_16 (word_param)

Loads a 16-bit -RD into the low half of the 32-bit "accumulator", zeros upper 16 bits of accumulator.

Load_32 (word_hi,word_lo)

Loads word_hi into upper 16 bits of accumulator, word_lo into Lower 16 bits.

Low_16

Returns the lower 16 bits of the accumulator, bits 0 through 15.

Mid_16

Returns the middle 16 bits of the accumulator, bits 8 through 23.

High_16

Returns the upper 16 bits of the accumulator, bits 16 through 31.

Mul_16 (word_param)

Multiplies the 32-bit accumulator by the 16-bit word supplied, result left in accumulator.

Div_16 (word_param)

Divides the 32-bit accumulator by the 16-bit word supplied, result left in accumulator.

Add_16 (word_param)

Adds the 16-bit word supplied to the 32-bit accumulator.

Sub_16 (word_param)

Similar to Add_16 but for subtraction.

Add_32 (word_hi,word_lo)

Forms a 32-bit value for word_hi and word_lo and adds it to the accumulator.

Sub_32 (word_hi,word_lo)

Similar to Add_32 but for subtraction.

APPLICATION

Typical applications have 16-bit "input" values and produce 16-bit "output" values, but require 32-bit values for intermediate results. An example would be reading a 12-bit A/D, performing some gain and offset calculation on the raw A/D data to produce a calibrated 16 bit result. Doing this is a simple task with this math package.

```
CALL Load_16(AD_value);
```

```
CALL Add_16 (offset_value);
```

```
CALL Mul_16 (gain_factor);
```

```
/* gain is in units of 1/256 */
```

```
result = Mid_16;
```

In this example the accumulator was loaded with the raw A/D value and then the offset was applied. The gain_factor was "pre-multiplied" by 256 (8 bits), giving it a granularity of 1/256. The result was extracted from the "middle" 16 bits of the accumulator (bits 8 to 23) to account for the scaling factor of 256 introduced in the multiply step.

The package requires about 384 bytes of ROM and 30 bytes of RAM. Individual routines can be deleted to conserve RAM if they are not used.

CODE SOURCE LISTINGS

CODE SOURCE LISTINGS

```

NAME      Math_32_Module
;
PUBLIC    Load_16, ?Load_16?byte
PUBLIC    Load_32, ?Load_32?byte
PUBLIC    Mul_16, ?Mul_16?byte
PUBLIC    Div_16, ?Div_16?byte
PUBLIC    Add_16, ?Add_16?byte
PUBLIC    Sub_16, ?Sub_16?byte
PUBLIC    Add_32, ?Add_32?byte
PUBLIC    Sub_32, ?Sub_32?byte
PUBLIC    Low_16, Mid_16, High_16
;
Math_32_Data SEGMENT DATA
Math_32_Code SEGMENT CODE
;
RSEG      Math_32_Data
?Load_16?byte: DS 2
?Load_32?byte: DS 4
?Mul_16?byte: DS 2
?Div_16?byte: DS 2
?Add_16?byte: DS 2
?Sub_16?byte: DS 2
?Add_32?byte: DS 4
?Sub_32?byte: DS 4
OP_0: DS 1
OP_1: DS 1
OP_2: DS 1
OP_3: DS 1
TMP_0: DS 1
TMP_1: DS 1
TMP_2: DS 1
TMP_3: DS 1
;
RSEG      Math_32_Code

```

```

Load_16:
;Load the lower 16 bits of the OP registers with the value supplied
MOV     OP_3,#0
MOV     OP_2,#0
MOV     OP_1,?Load_16?byte
MOV     OP_0,?Load_16?byte + 1
RET

Load_32:
;Load all the OP registers with the value supplied
MOV     OP_3,?Load_32?byte
MOV     OP_2,?Load_32?byte + 1
MOV     OP_1,?Load_32?byte + 2
MOV     OP_0,?Load_32?byte + 3
RET

Low_16:
;Return the lower 16 bits of the OP registers
MOV     R6,OP_1
MOV     R7,OP_0
RET

Mid_16:
;Return the middle 16 bits of the OP registers
MOV     R6,OP_2
MOV     R7,OP_1
RET

High_16:
;Return the high 16 bits of the OP registers
MOV     R6,OP_3
MOV     R7,OP_2
RET

Add_16:
;Add the 16 bits supplied by the caller to the OP registers
CLR     C
MOV     A,OP_0
ADDC    A,?Add_16?byte + 1 ;low byte first
MOV     OP_0,A
MOV     A,OP_1
ADDC    A,?Add_16?byte ;high byte + carry
MOV     OP_1,A
MOV     A,OP_2
ADDC    A,#0 ;propagate carry only
MOV     OP_2,A
MOV     A,OP_3
ADDC    A,#0 ;propagate carry only
MOV     OP_3,A
RET

```

270530-2

```

Add_32:
;Add the 32 bits supplied by the caller to the OP registers
CLR C
MOV A,OP_0
ADDC A,?Add_32?byte + 3 ;lowest byte first
MOV OP_0,A
MOV A,OP_1
ADDC A,?Add_32?byte + 2 ;mid-lowest byte + carry
MOV OP_1,A
MOV A,OP_2
ADDC A,?Add_32?byte + 1 ;mid-highest byte + carry
MOV OP_2,A
MOV A,OP_3
ADDC A,?Add_32?byte ;highest byte + carry
MOV OP_3,A
RET

Sub_16:
;Subtract the 16 bits supplied by the caller from the OP registers
CLR C
MOV A,OP_0
SUBB A,?Sub_16?byte + 1 ;low byte first
MOV OP_0,A
MOV A,OP_1
SUBB A,?Sub_16?byte ;high byte + carry
MOV OP_1,A
MOV A,OP_2
SUBB A,#0 ;propagate carry only
MOV OP_2,A
MOV A,OP_3
SUBB A,#0 ;propagate carry only
MOV OP_3,A
RET

Sub_32:
;Subtract the 32 bits supplied by the caller from the OP registers
CLR C
MOV A,OP_0
SUBB A,?Sub_32?byte + 3 ;lowest byte first
MOV OP_0,A
MOV A,OP_1
SUBB A,?Sub_32?byte + 2 ;mid-lowest byte + carry
MOV OP_1,A
MOV A,OP_2
SUBB A,?Sub_32?byte + 1 ;mid-highest byte + carry
MOV OP_2,A
MOV A,OP_3
SUBB A,?Sub_32?byte ;highest byte + carry
MOV OP_3,A
RET

```

270530-3


```

Mul_16:
;Multiply the 32 bit OP with the 16 value supplied
MOV     TMP_3,#0      ;clear out upper 16 bits
MOV     TMP_2,#0
;Generate the lowest byte of the result
MOV     B,OP_0
MOV     A,?Mul_16?byte+1
MUL     AB
MOV     TMP_0,A        ;low-order result
MOV     TMP_1,B        ;high order result
;Now generate the next higher order byte
MOV     B,OP_1
MOV     A,?Mul_16?byte+1
MUL     AB
ADD     A,TMP_1        ;low-order result
MOV     TMP_1,A        ; save
MOV     A,B            ; get high-order result
ADDC    A,TMP_2        ; include carry from previous operation
MOV     TMP_2,A        ; save
JNC     Mul_loop1
INC     TMP_3          ; propagate carry into TMP_3

Mul_loop1:
MOV     B,OP_0
MOV     A,?Mul_16?byte
MUL     AB
ADD     A,TMP_1        ;low-order result
MOV     TMP_1,A        ; save
MOV     A,B            ; get high-order result
ADDC    A,TMP_2        ; include carry from previous operation
MOV     TMP_2,A        ; save
JNC     Mul_loop2
INC     TMP_3          ; propagate carry into TMP_3

Mul_loop2:
; Now start working on the 3rd byte
MOV     B,OP_2
MOV     A,?Mul_16?byte+1
MUL     AB
ADD     A,TMP_2        ;low-order result
MOV     TMP_2,A        ; save
MOV     A,B            ; get high-order result
ADDC    A,TMP_3        ; include carry from previous operation
MOV     TMP_3,A        ; save
; Now the other half
MOV     B,OP_1
MOV     A,?Mul_16?byte
MUL     AB
ADD     A,TMP_2        ;low-order result
MOV     TMP_2,A        ; save
MOV     A,B            ; get high-order result
ADDC    A,TMP_3        ; include carry from previous operation
MOV     TMP_3,A        ; save
; Now finish off the highest order byte
MOV     B,OP_3
MOV     A,?Mul_16?byte+1

```

270530-4

```

MUL    AB
ADD    A,TMP_3      ;low-order result
MOV    TMP_3,A      ; save
; Forget about the high-order result, this is only 32 bit math!
MOV    B,OP_2
MOV    A,?Mul_16?byte
MUL    AB
ADD    A,TMP_3      ;low-order result
MOV    TMP_3,A      ; save
; Now we are all done, move the TMP values back into OP
MOV    OP_0,TMP_0
MOV    OP_1,TMP_1
MOV    OP_2,TMP_2
MOV    OP_3,TMP_3
RET

```

270530-5

```

; Now start working on the 1st byte
MOV    B,OP_0
MOV    A,?Mul_16?byte-1
MUL    AB
ADD    A,TMP_1
MOV    TMP_1,A
; save
; get high-order result
; include carry from previous operation
MOV    A,TMP_1
ADD    A,TMP_2
MOV    TMP_2,A
; save
; propagate carry into TMP_3
INC    TMP_3
; Now start working on the 2nd byte
MOV    B,OP_1
MOV    A,?Mul_16?byte-1
MUL    AB
ADD    A,TMP_2
MOV    TMP_2,A
; save
; get high-order result
; include carry from previous operation
MOV    A,TMP_2
ADD    A,TMP_3
MOV    TMP_3,A
; save
; propagate carry into TMP_4
INC    TMP_4
; Now start working on the 3rd byte
MOV    B,OP_2
MOV    A,?Mul_16?byte-1
MUL    AB
ADD    A,TMP_3
MOV    TMP_3,A
; save
; get high-order result
; include carry from previous operation
MOV    A,TMP_3
ADD    A,TMP_4
MOV    TMP_4,A
; save
; propagate carry into TMP_5
INC    TMP_5
; Now finish off the highest order byte
MOV    B,OP_3
MOV    A,?Mul_16?byte-1
MUL    AB
ADD    A,TMP_4
MOV    TMP_4,A
; save
; get high-order result
; include carry from previous operation
MOV    A,TMP_4
ADD    A,TMP_5
MOV    TMP_5,A
; save
; propagate carry into TMP_6
INC    TMP_6

```

270530-4

```

Div_16:
;This divides the 32 bit OP register by the value supplied
MOV R7,#0
MOV R6,#0 ;zero out partial remainder
MOV TMP_0,#0
MOV TMP_1,#0
MOV TMP_2,#0
MOV TMP_3,#0
MOV R1,?Div_16?byte ;load divisor
MOV R0,?Div_16?byte+1
MOV R5,#32 ;loop count
;This begins the loop
Div_loop:
CALL Shift_D ;shift the dividend and return MSB in C
MOV A,R6 ;shift carry into LSB of partial remainder
RLC A
MOV R6,A
MOV A,R7
RLC A
MOV R7,A
;now test to see if R7:R6 >= R1:R0
JC Can_sub ;Carry out of R7 shift means R7:R6 > R1:R0
CLR C
MOV A,R7 ;subtract R1 from R7 to see if R1 < R7
SUBB A,R1 ; A = R7 - R1, carry set if R7 < R1
JC Cant_sub
;at this point R7>R1 or R7=R1
JNZ Can_sub ;jump if R7>R1
;if R7 = R1, test for R6>=R0
CLR C
MOV A,R6
SUBB A,R0 ; A = R6 - R0, carry set if R6 < R0
JC Cant_sub
Can_sub:
;subtract the divisor from the partial remainder
CLR C
MOV A,R6
SUBB A,R0 ; A = R6 - R0
MOV R6,A
MOV A,R7
SUBB A,R1 ; A = R7 - R1 - Borrow
MOV R7,A
SETB C ; shift a 1 into the quotient
JMP Quot
Cant_sub:
;shift a 0 into the quotient
CLR C
Quot:
;shift the carry bit into the quotient
CALL Shift_Q
; Test for completion
DJNZ R5,Div_loop
; Now we are all done, move the TMP values back into OP
MOV OP_0,TMP_0
MOV OP_1,TMP_1

```

270530-6

```

MOV    OP_2,TMP_2
MOV    OP_3,TMP_3
RET

```

Shift D:

```

;shift the dividend one bit to the left and return the MSB in C
CLR    C
MOV    A,OP_0
RLC    A
MOV    OP_0,A
MOV    A,OP_1
RLC    A
MOV    OP_1,A
MOV    A,OP_2
RLC    A
MOV    OP_2,A
MOV    A,OP_3
RLC    A
MOV    OP_3,A
RET

```

Shift Q:

```

;shift the quotient one bit to the left and shift the C into LSB
MOV    A,TMP_0
RLC    A
MOV    TMP_0,A
MOV    A,TMP_1
RLC    A
MOV    TMP_1,A
MOV    A,TMP_2
RLC    A
MOV    TMP_2,A
MOV    A,TMP_3
RLC    A
MOV    TMP_3,A
RET

```

END

```

; This divides the 32 bit OP register by the value supplied
; into the partial remainder
MOV    R7,R0
MOV    R6,R0
MOV    TMP_0,R0
MOV    TMP_1,R0
MOV    TMP_2,R0
MOV    TMP_3,R0
MOV    R1,TMP_0
MOV    R0,TMP_0
MOV    R2,$32
; This begins the loop
DivLoop:
CALL    Shift_D
; Shift the dividend and return MSB in C
; Shift carry into LSB of partial remainder
MOV    A,R6
RLC    A
MOV    R6,A
MOV    A,R7
RLC    A
MOV    R7,A
; Now test to see if R7:R6 >= R1:R0
JC      CanSub
; Carry out of R7 shift means R7:R6 > R1:R0
CLR    C
MOV    A,R7
SUBB   A,R1
; A = R7 - R1, carry set if R7 < R1
JC      CanSub
; At this point R7:R1 or R7:R0
; Can sub
; If R7 = R1, test for R6 >= R0
CLR    C
MOV    A,R6
SUBB   A,R0
; A = R6 - R0, carry set if R6 < R0
JC      CanSub
; Subtract the divisor from the partial remainder
CLR    C
MOV    A,R6
SUBB   A,R0
; A = R6 - R0
MOV    R6,A
SUBB   A,R1
; A = R7 - R1 - borrow
MOV    R7,A
; Shift a 1 into the quotient
SETB   C
JMP     Quot
CanSub:
; Shift the carry bit into the quotient
CLR    C
CALL    Shift_Q
; Test for completion
DUMB   R2,DivLoop
; Now we are all done, move the TMP values back into OP
MOV    OP_0,TMP_0
MOV    OP_1,TMP_1

```

270530-7

270530-8



PAGE	CONTENTS
2-181	INTRODUCTION
2-181	80C80 MAILBOX
2-182	80C31 MAILBOX CONTROLLER
2-183	Block Diagram
2-184	80C80 "Back to Back Register"
2-185	80C31 "Mailbox Controller"
2-186	80C80 Register ADF
2-187	80C31 Address ADF

DESIGNING A MAILBOX
MEMORY FOR TWO 80C31
MICROCONTROLLERS
USING EPLDs

October 1989

2

Designing a Mailbox Memory for Two 80C31 Microcontrollers Using EPLDs

K. WEIGL & J. STAHL
INTEL CORPORATION
MUNICH, GERMANY

Order Number: 292016-004

**DESIGNING A MAILBOX
MEMORY FOR TWO 80C31
MICROCONTROLLERS
USING EPLDs**

CONTENTS

PAGE

INTRODUCTION	2-181
5C060 MAILBOX	2-181
5C032 MAILBOX CONTROLLER	2-182
Block Diagram	2-183
5C060 "Back to Back Register"	2-184
5C032 "Mailbox Controller"	2-185
5C060 Register ADF	2-186
5C032 Arbiter ADF	2-187

October 1989

2

Designing a Mailbox Memory for
Two 80C31 Microcontrollers Using
EPLDs

K. WEIGL & J. STAHL
INTEL CORPORATION
MUNICH, GERMANY

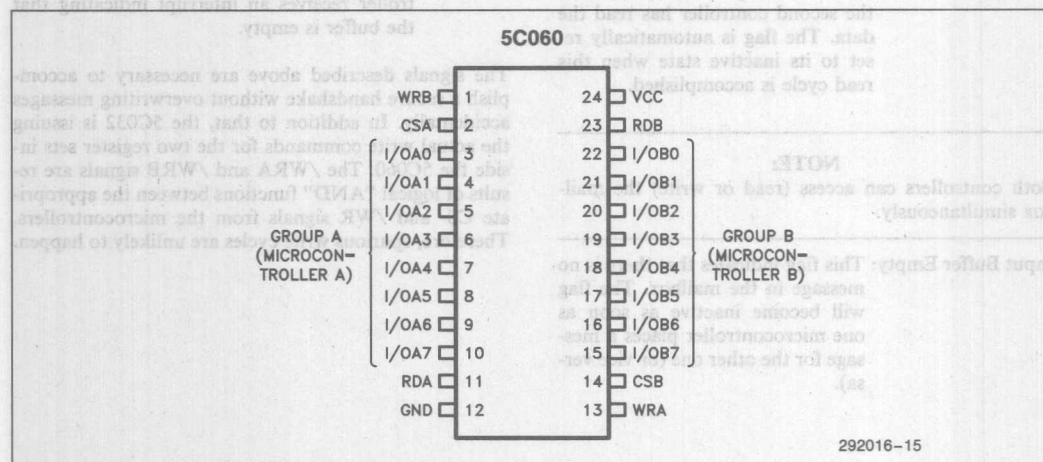
Order Number: 385018-004

INTRODUCTION

Very often, complex systems involve two or more microcontrollers to fulfill the requirements defined by a given objective. Since the nature of microcontrollers does not allow for easy dual-port memory design (no "READY" input; no "HOLD/HLDA" interface; port-oriented I/O etc.), design engineers are faced with the problem of interchanging information (data and status) between those microcontrollers. This application brief describes the design of a mailbox for exchanging information between two 80C31s, using a 5C060 EPLD as a "back-to-back" register, and a 5C032 EPLD as an arbitration vehicle to control the actions of the CPUs.

THE 5C060 MAILBOX

In this application, the 16 macrocells of the 5C060 are grouped into two sets of 8 so called "ROIF" (register output with input feedback) primitives to implement the two 8 bit bus interfaces needed. The grouping is done according to the following picture.



The 5C060 allows for independent clocking of 8 macrocells on each side of the chip, the two clock inputs are used to clock data from the microcontroller bus into the chip. To read the data written into the mailbox by one of the controllers, the RDA- (controller A is reading) or RDB- (controller B is reading) line must be pulled low by activating the read command (/RD). In order to avoid spurious read-cycles, the /RD commands from both microcontrollers are logically "ORed" together with an active high CS-signal (Chip Select) inside the 5C060. The CS-signal for both ports is derived from address line A15. Therefore, whenever A15 becomes a logic "1" (true), the mailbox is activated and ready to take or submit data.

Address range for the mailbox: F000 Hex to FFFF Hex
(Upper 12 kbyte)

THE 5C032 "MAILBOX CONTROLLER"

To keep the two microcontrollers informed about the status of their mailbox, the 5C032 is programmed to supply the following signals to both controllers:

/OBFA: "OUTPUT BUFFER FULL" FOR MC A

/OBFB: "OUTPUT BUFFER FULL" FOR MC B

/IBEA: "INPUT BUFFER EMPTY" FOR MC A

/IBEB: "INPUT BUFFER EMPTY" FOR MC B

/INTA: INTERRUPT TO MC A

/INTB: INTERRUPT TO MC B

The next section will discuss the meanings of these signals in more detail.

Output Buffer Full: This flag is set whenever the controller writes into its own output buffer. The flag remains valid, until the second controller has read the data. The flag is automatically reset to its inactive state when this read cycle is accomplished.

NOTE:

Both controllers can access (read or write) the mailbox simultaneously.

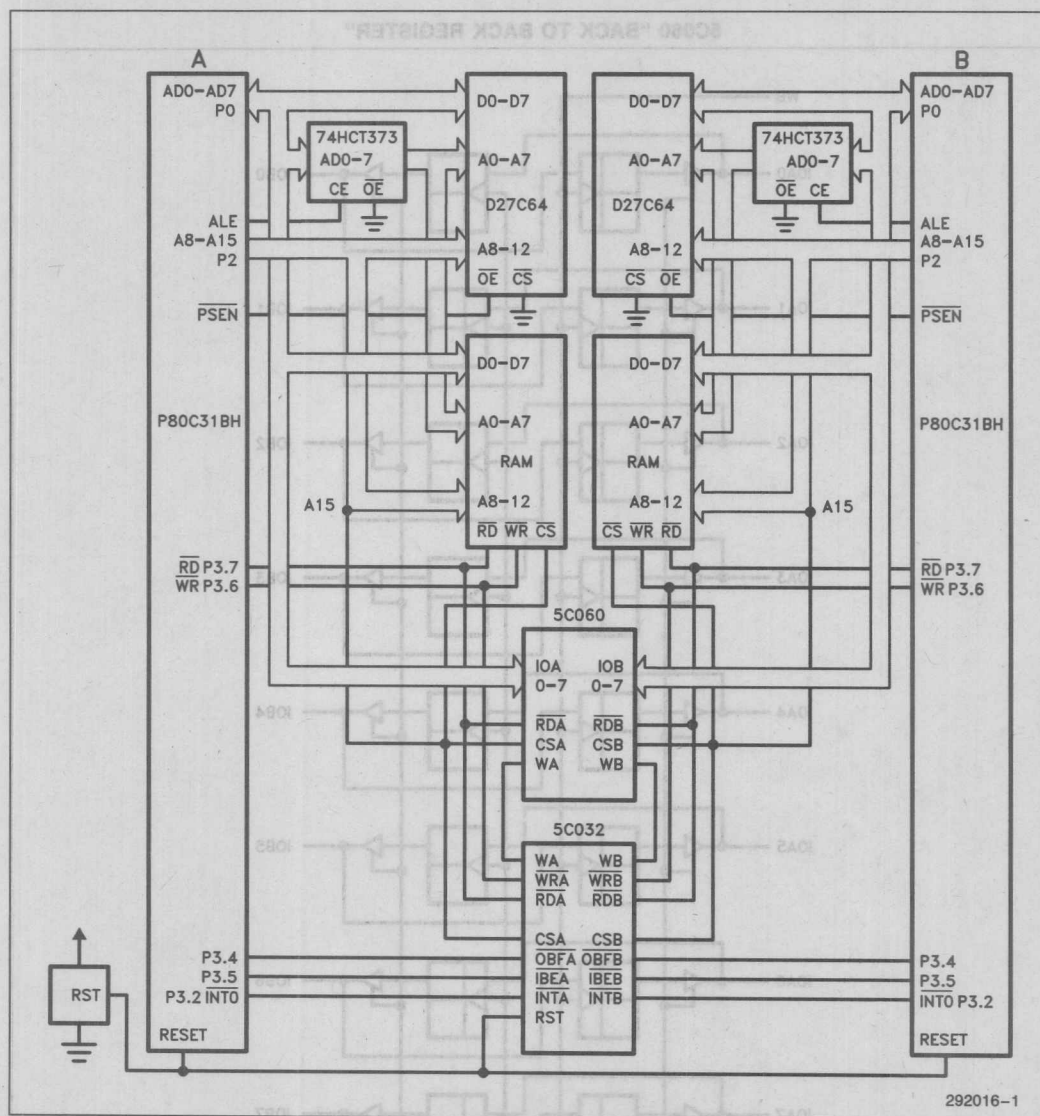
Input Buffer Empty: This flag indicates that there is no message in the mailbox. The flag will become inactive as soon as one microcontroller places a message for the other one (or vice versa).

Example: /IBEA remains "LOW" until microcontroller B places a message for controller A. /IBEA will go "HIGH" as soon as controller B has accomplished its write cycle, and will not go "LOW" again until microcontroller A has read the message.

Interrupt: The 5C032 is programmed to supply interrupts to both microcontrollers involved, on one of the following events.

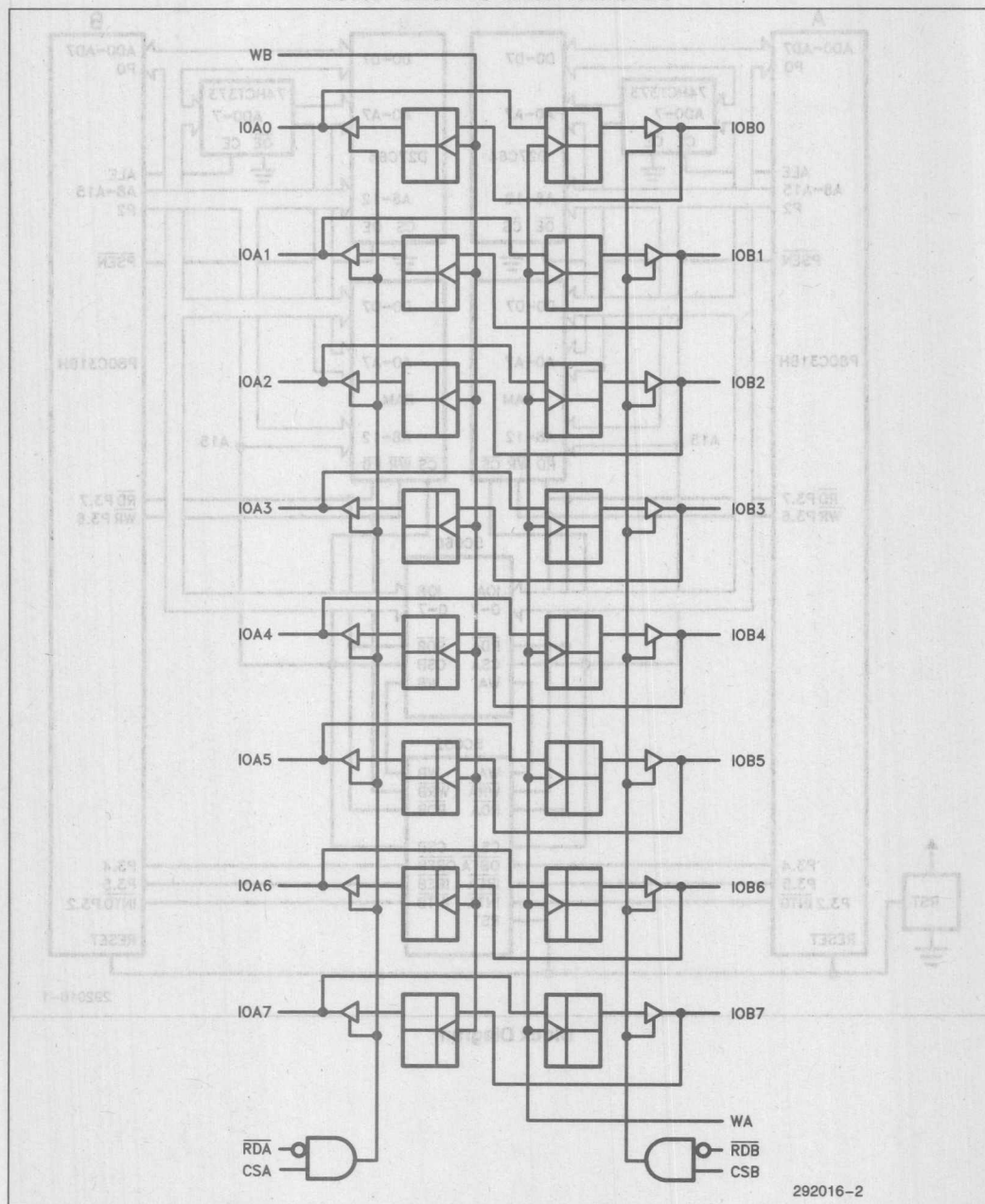
1. The /OBF flag of the opposite microcontroller becomes active; e.g. if controller A is placing a message for controller B, controller B receives an interrupt the same time as /OBFA becomes valid or vice versa.
2. The /IBE flag of the opposite microcontroller goes active, indicating that this controller has received the message; e.g. if controller B reads the message stored by controller A, its /IBEB flag goes active and controller receives an interrupt indicating that the buffer is empty.

The signals described above are necessary to accomplish a secure handshake without overwriting messages accidentally. In addition to that, the 5C032 is issuing the actual write commands for the two register sets inside the 5C060. The /WRA and /WRB signals are results of logical "AND" functions between the appropriate CS- and /WR signals from the microcontrollers. Therefore, spurious write cycles are unlikely to happen.

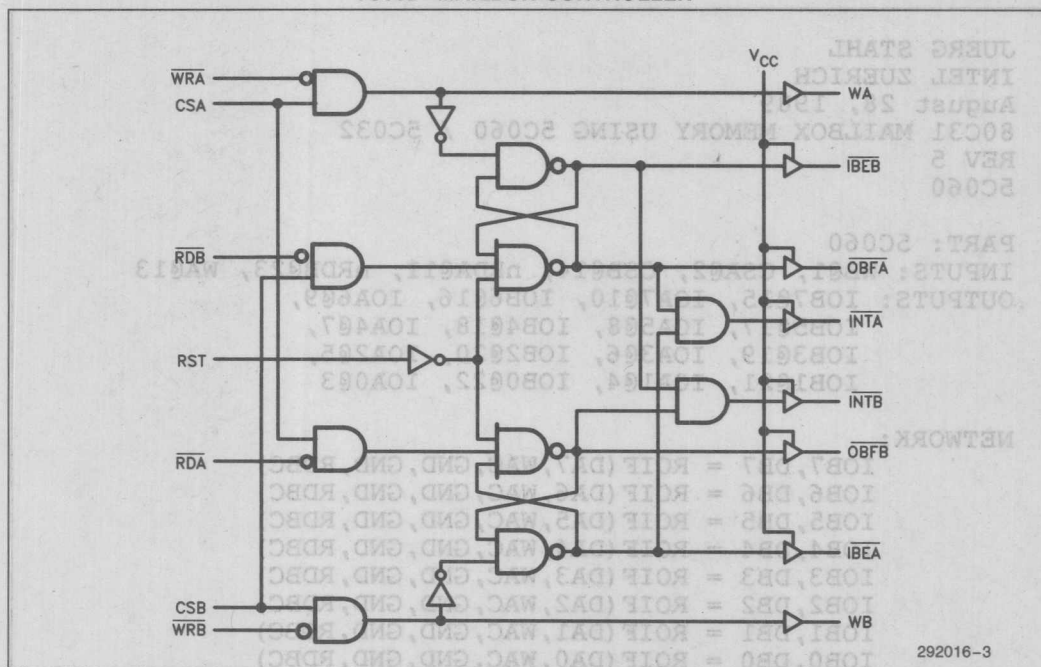


Block Diagram

5C060 "BACK TO BACK REGISTER"



5C032 "MAILBOX CONTROLLER"



2

5C060 REGISTER ADF

JUERG STAHL
INTEL ZUERICH
August 28, 1989
80C31 MAILBOX MEMORY USING 5C060 / 5C032
REV 5
5C060

PART: 5C060

INPUTS: WB@1, CSA@2, CSB@14, nRDA@11, nRDB@23, WA@13

OUTPUTS: IOB7@15, IOA7@10, IOB6@16, IOA6@9,
IOB5@17, IOA5@8, IOB4@18, IOA4@7,
IOB3@19, IOA3@6, IOB2@20, IOA2@5,
IOB1@21, IOA1@4, IOB0@22, IOA0@3

NETWORK:

IOB7,DB7 = ROIF (DA7,WAC,GND,GND,RDBC)
IOB6,DB6 = ROIF (DA6,WAC,GND,GND,RDBC)
IOB5,DB5 = ROIF (DA5,WAC,GND,GND,RDBC)
IOB4,DB4 = ROIF (DA4,WAC,GND,GND,RDBC)
IOB3,DB3 = ROIF (DA3,WAC,GND,GND,RDBC)
IOB2,DB2 = ROIF (DA2,WAC,GND,GND,RDBC)
IOB1,DB1 = ROIF (DA1,WAC,GND,GND,RDBC)
IOB0,DB0 = ROIF (DA0,WAC,GND,GND,RDBC)
IOA7,DA7 = ROIF (DB7,WBC,GND,GND,RDAC)
IOA6,DA6 = ROIF (DB6,WBC,GND,GND,RDAC)
IOA5,DA5 = ROIF (DB5,WBC,GND,GND,RDAC)
IOA4,DA4 = ROIF (DB4,WBC,GND,GND,RDAC)
IOA3,DA3 = ROIF (DB3,WBC,GND,GND,RDAC)
IOA2,DA2 = ROIF (DB2,WBC,GND,GND,RDAC)
IOA1,DA1 = ROIF (DB1,WBC,GND,GND,RDAC)
IOA0,DA0 = ROIF (DB0,WBC,GND,GND,RDAC)

WAC = INP (WA)
WBC = INP (WB)
CSB = INP (CSB)
CSA = INP (CSA)
nRDB = INP (nRDB)
nRDA = INP (nRDA)

EQUATIONS:

RDBC = CSB * !nRDB;

RDAC = CSA * !nRDA;

END\$

5C032 ARBITER ADF

JUERG STAHL
 INTEL ZUERICH
 August 28, 1989
 80C31 MAILBOX MEMORY USING 5C060 / 5C032
 REV 5
 5C032

PART: 5C032
 INPUTS: RST, nWRA, nRDB, CSA, nRDA, nWRB, CSB
 OUTPUTS: WA, nOBFA, nIBEB, nINTA, nINTB, nOBFB, nIBEA, WB

NETWORK:

nWRA = INP (nWRA)
 nRDA = INP (nRDA)
 CSA = INP (CSA)
 nWRB = INP (nWRB)
 nRDB = INP (nRDB)
 CSB = INP (CSB)
 RST = INP (RST)
 WA = CONF (WAd, VCC)
 WB = CONF (WBd, VCC)
 nOBFA, nOBFA = COIF (nOBFAAd, VCC)
 nOBFB, nOBFB = COIF (nOBFBd, VCC)
 nIBEA, nIBEA = COIF (nIBEAAd, VCC)
 nIBEB, nIBEB = COIF (nIBEBd, VCC)
 nINTA = CONF (nINTAd, VCC)
 nINTB = CONF (nINTBd, VCC)

EQUATIONS:

nINTBd = nOBFB * nIBEB;
 nINTAd = nOBFA * nIBEA;
 nOBFBd = !(!(nRDA * CSA) * nIBEA * !RST);
 nOBFAAd = !(!(nRDB * CSB) * nIBEB * !RST);
 nIBEBd = !(!(CSA * !nWRA) * nOBFA);
 nIBEAAd = !(!(CSB * !nWRB) * nOBFB);
 WAd = CSA * !nWRA;
 WBd = CSB * !nWRB;

END\$



APPLICATION NOTE

AP-252

80C31 MAILBOX MEMORY USING 80C32 \ 80C32
REV 5
50032
PART: 50032
INPUTS: RST, nWRA, nRDB, CSA, nRDA, nWRB, CSB
OUTPUTS: WA, nOBEA, nIBEb, nINTA, nINTB, nOBEb, nIBEa, WB

March 1985

Designing With The 80C51BH

EQUATIONS:

WbD = CSB * !nWRB;
WbA = CSA * !nWRA;
nIBEaD = !((CSB * !nWRB) * nOBEb);
nIBEbD = !((CSA * !nWRA) * nOBEa);
nOBEaD = !((!nRDB * CSB) * nIBEb * !RST);
nOBEbD = !((!nRDA * CSA) * nIBEa * !RST);
nINTaD = nOBEa * nIBEa;
nINTbD = nOBEb * nIBEb;

ENDS

TOM WILLIAMSON
MCO APPLICATIONS ENGINEER

582016-9

Order Number: 270068-002

DESIGNING WITH THE 80C51BH

Lower operating voltages are easier to obtain with the p-well structure than with the n-well structure. But the p-well structure does not easily adapt to an EPROM which would be pin-for-pin compatible with HMOS. On the other hand the n-well structure can be based on the solidly founded HMOS process, in which nFETs are built into a p-type substrate. This allows somewhat more than half of the transistor in a CMOS chip to be constructed by processes that are already well characterized.

Currently Intel's CMOS microcontrollers and microprocessors are p-well devices.

Further discussion of the CMOS technology is provided in References 1 and 2 (which are reprinted in the Microcontroller Handbook).

THE MCS-51 FAMILY IN CMOS

The 80C51BH is the CMOS version of Intel's original 8051. The 80C31BH is the ROMless 80C51BH equivalent to the 8051. These CMOS devices are pin-for-pin identical with their HMOS counterparts, except that they have two added features for reduced power. These are the Idle and Power Down modes of operation.

In most cases, an 80C51BH can directly replace the 8051 in existing applications. It can execute the same code at the same speed, accept signals from the same sources and drive the same loads. However, the 80C51BH covers a wider range of speeds, will meet CMOS logic levels to CMOS loads, and will draw about 1/10 the current of an 8051 (and less yet in the reduced power modes). Interchangeability between the HMOS and CMOS devices is discussed in more detail in the final section of this Application Note.

It should be noted that the 80C51BH CPU is not static. That means if the clock frequency is too low, the CPU might forget what it was doing. This is because the circuitry uses a number of dynamic nodes. A dynamic node is one that uses the node-to-ground capacitance to form a temporary storage cell. Dynamic nodes are used to reduce the transistor count, and hence the chip area, thus to produce a more economical device.

This is not to say that the on-chip RAM in CMOS microcontrollers is dynamic. It's not. It's the CPU that is dynamic, and that is what imposes the minimum clock frequency specification.

CONTENTS

CMOS EVOLVES	2-190
WHAT IS CHMOS?	2-190
THE MCS-51 FAMILY IN CHMOS	2-190
LATCHUP	2-191
LOGIC LEVELS AND INTERFACING PROBLEMS	2-191
NOISE CONSIDERATIONS	2-191
UNUSED PINS	2-192
PULLUP RESISTORS	2-193
PULLDOWN RESISTORS	2-193
DRIVE CAPABILITY OF THE INTERNAL PULLUPS	2-194
POWER CONSUMPTION	2-195
Idle	2-196
Power Down	2-196
USING THE POWER DOWN MODE	2-197
USING POWER MOSFETs TO CONTROL V _{CC}	2-198
BATTERY BACKUP SYSTEMS	2-198
POWER SWITCHOVER CIRCUITS	2-200
80C31BH + CHMOS EPROM	2-201
SCANNING A KEYBOARD	2-202
DRIVING AN LCD	2-205
Using an LCD Driver	2-206
RESONANT TRANSDUCERS	2-207
Frequency Measurements	2-207
Period Measurements	2-209
Pulse Width Measurements	2-210
HMOS/CHMOS INTERCHANGEABILITY	2-210
External Clock Drive	2-210
Unused Pins	2-211
Logic Levels	2-211
Idle and Power Down	2-211
REFERENCES	2-212

CMOS EVOLVES

The original CMOS logic families were the 4000-series and the 74C-series circuits. The 74C-series circuits are functional equivalents to the corresponding numbered 74-series TTL circuits, but have CMOS logic levels and retain the other well known characteristics of CMOS logic.

These characteristics are: low power consumption, high noise immunity, and slow speed. The low power consumption is inherent to the nature of the CMOS circuit. The noise immunity is due partly to the CMOS logic levels, and partly to the slowness of the circuits. The slow speed is due to the technology used to construct the transistors in the circuit.

The technology used is called metal-gate CMOS, because the transistor gates are formed by metal deposition. More importantly, the gates are formed after the drain and source regions have been defined, and must overlap the source and drain somewhat to allow for alignment tolerances. This overlap plus the relatively large size of the transistors themselves result in high electrode capacitance, and that is what limits the speed of the circuit.

High speed CMOS became feasible with the development of the self-aligning silicon gate technology. In this process polysilicon gates are deposited before the source and drain regions are defined. Then the source and drain regions are formed by ion implantation using the gate itself as a mask for the implantation. This eliminates most of the overlap capacitance. In addition, the process allows smaller transistors. The result is a significant increase in circuit speed. The 74HC-series of CMOS logic circuits is based on this technology, and has speeds comparable to LS TTL, which is to say about 10 times faster than the 74C-series circuits.

The size reduction that contributes to the higher speed also demands an accompanying reduction in the maximum supply voltage. High-speed CMOS is generally limited to 6V.

WHAT IS CHMOS?

CHMOS is the name given to Intel's high-speed CMOS processes. There are two CHMOS processes, one based on an n-well structure and one based on a p-well structure. In the n-well structure, n-type wells are diffused into a p-type substrate. Then the n-channel transistors (nFETs) are built into the substrate and pFETs are built into the n-wells. In the p-well structure, p-type wells are diffused into an n-type substrate. Then the nFETs are built into the wells and pFETs, into the

substrate. Both processes have their advantages and disadvantages, which are largely transparent to the user.

Lower operating voltages are easier to obtain with the p-well structure than with the n-well structure. But the p-well structure does not easily adapt to an EPROM which would be pin-for-pin compatible with HMOS EPROMs. On the other hand the n-well structure can be based on the solidly founded HMOS process, in which nFETs are built into a p-type substrate. This allows somewhat more than half of the transistors in a CHMOS chip to be constructed by processes that are already well characterized.

Currently Intel's CHMOS microcontrollers and memory products are n-well devices, whereas CHMOS microprocessors are p-well devices.

Further discussion of the CHMOS technology is provided in References 1 and 2 (which are reprinted in the Microcontroller Handbook).

THE MCS®-51 FAMILY IN CHMOS

The 80C51BH is the CHMOS version of Intel's original 8051. The 80C31BH is the ROMless 80C51BH, equivalent to the 8031. These CHMOS devices are architecturally identical with their HMOS counterparts, except that they have two added features for reduced power. These are the Idle and Power Down modes of operation.

In most cases, an 80C51BH can directly replace the 8051 in existing applications. It can execute the same code at the same speed, accept signals from the same sources, and drive the same loads. However, the 80C51BH covers a wider range of speeds, will emit CMOS logic levels to CMOS loads, and will draw about 1/10 the current of an 8051 (and less yet in the reduced power modes). Interchangeability between the HMOS and CHMOS devices is discussed in more detail in the final section of this Application Note.

It should be noted that the 80C51BH CPU is not static. That means if the clock frequency is too low, the CPU might forget what it was doing. This is because the circuitry uses a number of dynamic nodes. A dynamic node is one that uses the node-to-ground capacitance to form a temporary storage cell. Dynamic nodes are used to reduce the transistor count, and hence the chip area, thus to produce a more economical device.

This is not to say that the on-chip RAM in CHMOS microcontrollers is dynamic. It's not. It's the CPU that is dynamic, and that is what imposes the minimum clock frequency specification.

LATCHUP

Latchup is an SCR-type turn-on phenomenon that is the traditional nemesis of CMOS systems. The substrate, the wells, and the transistors form parasitic pnpn structures within the device. These parasitic structures turn on like an SCR if a sufficient amount of forward current is driven through one of the junctions. From the circuit designer's point of view it can happen whenever an input or output pin is externally driven a diode drop above V_{CC} or below V_{SS} , by a source that is capable of supplying the required trigger current.

However much of a problem latchup has been in the past, it is good to know that in most recently developed CMOS devices, and specifically in CHMOS devices, the current required to trigger latchup is typically well over 100 mA. The 80C51BH is virtually immune to latchup. (References 1 and 2 present a discussion of the latchup mechanisms and the steps that are taken on the chip to guard against it.) Modern CMOS is not absolutely immune to latchup, but with trigger currents in the hundreds of mA, latchup is certainly a lot easier to avoid than it once was.

A careless power-up sequence might trigger a latchup in the older CMOS families, but it's unlikely to be a major problem in high-speed CMOS or in CHMOS. There is still some risk incurred in inserting or removing chips or boards in a CMOS system while the power is on. Also, severe transients, such as inductive kicks or momentary short-circuits, can exceed the trigger current for latchup.

For applications in which some latchup risk seems unavoidable, you can put a small resistor (100 Ω or so) in series with signal lines to ensure that the trigger current will never be reached. This also helps to control overshoot and RFI.

LOGIC LEVELS AND INTERFACING PROBLEMS

CMOS logic levels differ from TTL levels in two ways.

Logic State	$V_{CC} = 5V$				
	74HC	74HCT	LS TTL	8051	80C51BH
V_{IH}	3.5V	2.0V	2.0V	2.0V	1.9V
V_{IL}	1.0V	0.8V	0.8V	0.8V	0.9V
V_{OH}	4.9V	4.9V	2.7V	2.4V	4.5V
V_{OL}	0.1V	0.1V	0.5V	0.45V	0.45V

Figure 1. Logic Level Comparison. (Output voltage levels depend on load current. Data sheets list guaranteed output levels for several load currents. The output levels listed here are for minimum loading.)

First, for equal supply voltages, CMOS gives (and requires) a higher "logic 1" level than TTL. Secondly, CMOS logic levels are V_{CC} (or V_{DD}) dependent, whereas guaranteed TTL logic levels are fixed when V_{CC} is within TTL specs.

Standard 74HC logic levels are as follows:

$$\begin{aligned} V_{IHMIN} &= 70\% \text{ of } V_{CC} \\ V_{ILMAX} &= 20\% \text{ of } V_{CC} \\ V_{OHMIN} &= V_{CC} - 0.1V, |I_{OH}| \leq 20 \mu A \\ V_{OLMAX} &= 0.1V, |I_{OL}| \leq 20 \mu A \end{aligned}$$

Figure 1 compares 74HC, LS TTL, and 74HCT logic levels with those of the HMOS 8051 and the CHMOS 80C51BH for $V_{CC} = 5V$.

Output logic levels depend of course on load current, and are normally specified at several load currents. When CMOS and TTL are powered by the same V_{CC} , the logic levels guaranteed on the data sheets indicate that CMOS can drive TTL, but TTL can't drive CMOS. The incompatibility is that the TTL circuit's V_{OH} level is too low to reliably be recognized by the CMOS circuit as a valid V_{IH} .

Since HMOS circuits were designed to be TTL-compatible, they have the same incompatibility.

Fortunately, 74HCT-series circuits are available to ease these interfacing problems. They have TTL-compatible logic levels at the inputs and standard CMOS levels at the outputs.

The 80C51BH is designed to work with either TTL or CMOS. Therefore its logic levels are specified very much like 74HCT circuits. That is, its input logic levels are TTL-compatible, and its output characteristics are like standard high-speed CMOS.

NOISE CONSIDERATIONS

One of the major reasons for going to CMOS has traditionally been that CMOS is less susceptible to noise. As previously noted, its low susceptibility to noise is

partly due to superior noise margins, and partly due to its slow speed.

Noise margin is the difference between V_{OL} and V_{IL} , or between V_{OH} and V_{IH} . If V_{OH} from a driving circuit is 2.7V and V_{IH} to the driven circuit is 2.0V, then the driven circuit has 0.7V of noise margin at the logic high level. These kinds of comparisons show that an all-CMOS system has wider noise margins than an all-TTL system.

Figure 2 shows noise margins in CMOS and LS TTL systems when both have $V_{CC} = 5V$. It can be seen that CMOS/CMOS and CMOS/CHMOS systems have an edge over LS TTL in this respect.

Noise margins can be misleading, however, because they don't say how much noise energy it takes to induce in the circuit a noise voltage of sufficient amplitude to cause a logic error. This would involve consideration of the width of the noise pulse as compared with the circuit's response speed, and the impedance to ground from the point of noise introduction in the circuit.

When these considerations are included, it is seen that using the slower 74C- and 4000-series circuits with a 12 or 15V supply voltage does offer a truly improved level of noise immunity, but that high-speed CMOS at 5V is not significantly better than TTL.

One should not mistake the wider supply voltage tolerance of high-speed CMOS for V_{CC} glitch immunity. Supply voltage tolerance is a DC rating, not a glitch rating.

For any clocked CMOS, and most especially for VLSI CMOS, V_{CC} decoupling is critical. CHMOS draws

current in extremely sharp spikes at the clock edges. The VHF and UHF components of these spikes are not drawn from the power supply, but from the decoupling capacitor. If the decoupling circuit is not sufficiently low in inductance, V_{CC} will glitch at each clock edge. We suggest that a 0.1 μF decoupler cap be used in a minimum-inductance configuration with the microcontroller. A minimum-inductance configuration is one that minimizes the area of the loop formed by the chip (V_{CC} to V_{SS}), the traces to the decoupler cap, and the decoupler cap. PCB designers too often fail to understand that if the traces that connect the decoupler cap to the V_{CC} and V_{SS} pins aren't short and direct, the decoupler loses much of its effectiveness.

Overshoot and ringing in signal lines are potential sources of logic upsets. These can largely be controlled by circuit layout. Inserting small resistors (about 100 Ω) in series with signal lines that seem to need them will also help.

The sharp edges produced by high-speed CMOS can cause RFI problems. The severity of these problems is largely a function of the PCB layout. We don't mean to imply that all RFI problems can be solved by a better PCB layout. It may well be, for example, that in some RFI-sensitive designs high-speed CMOS is simply not the answer. But circuit layout is a critical factor in the noise performance of any electronic system, and more so in high-speed CMOS systems than others.

Circuit layout techniques for minimizing noise susceptibility and generation are discussed in References 3 through 6.

UNUSED PINS

CMOS input pins should not be left to float, but should always be pulled to one logic level or the other. If they float, they tend to float into the transition region between 0 and 1, where the pullup and pulldown devices in the input buffer are both conductive. This causes a significant increase in I_{CC} . A similar effect exists in HMOS circuits, but with less noticeable results.

In 80C51BH and 80C31BH designs, unused pins of Ports 1, 2, and 3 can be ignored, because they have internal pullups that will hold them at a valid Logic 1 level. Port 0 pins are different, however, in not having internal pullups (except during bus operations).

When the 80C51BH is in reset, the Port 0 pins are in a float state unless they are externally pulled up or down. If it's going to be held in reset for just a short time, the transient float state can probably be ignored. When it comes out of reset, the pins stay afloat unless

Interface	Noise Margin for $V_{CC} = 5V$	
	Logic Low $V_{IL} - V_{OL}$	Logic High $V_{OH} - V_{IH}$
74HC to 74HC	0.9V	1.4V
LSTTL to LSTTL	0.3V	0.7V
LSTTL to 74HCT	0.3V	0.7V
LSTTL to 80C51BH	0.3V	0.7V
74HC to 80C51BH	0.8V	3.0V
80C51BH to 74HC	0.8V	1.0V

Figure 2. Noise Margins for CMOS and LS TTL Circuits

they are externally pulled either up or down. Alternatively, the software can internally write 0s to whatever Port 0 pins may be unused.

The same considerations are applicable to the 80C31BH with regards to reset. But when the 80C31BH comes out of reset, it commences bus operations, during which the logic levels at the pins are always well defined as high or low.

Consider the 80C31BH in the Power Down or Idle modes, however. In those modes it is not fetching instructions, and the Port 0 pins will float if not externally pulled high or low. The choice of whether to pull them high or low is the designer's. Normally it is sufficient to pull them up to V_{CC} with 10k resistors. But if power is going to be removed from circuits that are connected to the bus, it will be advisable to pull the bus pins down (normally with 10k resistors). Considerations involved in selecting pullup and pulldown resistor values are as follows.

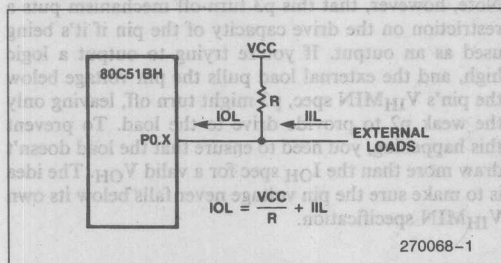


Figure 3a. Conditions defining the minimum value for R. P0.X is emitting a logic low. R must be large enough to not cause I_{OL} to exceed data sheet specifications.

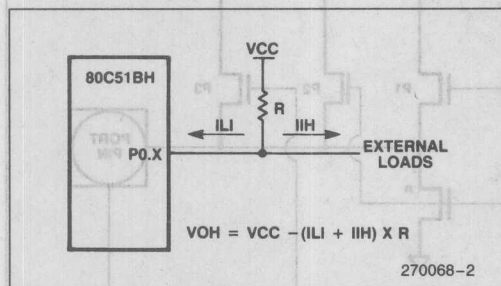


Figure 3b. Conditions defining the maximum value for R. P0.X is in a high impedance state. R must be small enough to keep V_{OH} acceptably high.

PULLUP RESISTORS

If a pullup resistor is to be used on a Port 0 pin, its minimum value is determined by I_{OL} requirements. If the pin is trying to emit a 0, then it will have to sink the current from the pullup resistor plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 3a, while maintaining a valid output low (V_{OL}). To guarantee that the pin voltage will not exceed 0.45V, the resistor should be selected so that I_{OL} doesn't exceed the value specified on the data sheet. In most CMOS applications, the minimum value would be about 2k Ω .

The maximum value you could use depends on how fast you want the pin to pull up after bus operations have ceased, and how high you want the V_{OH} level to be. The smaller the resistor the faster it pulls up. Its effect on the V_{OH} level is that $V_{OH} = V_{CC} - (I_{LI} + I_{IH}) \times R$. I_{LI} is the input leakage current to the Port 0 pin, and I_{IH} is the input high current to the external loads, as shown in Figure 3b. Normally V_{OH} can be expected to reach 0.9 V_{CC} if the pullup resistance does not exceed about 50k Ω .

Pulldown Resistors

If a pulldown resistor is to be used on a Port 0 pin, its minimum value is determined by V_{OH} requirements during bus operations, and its maximum value is in most cases determined by leakage current.

During bus operations the port uses internal pullups to emit 1s. The D.C. Characteristics in the data sheet list guaranteed V_{OH} levels for given I_{OH} currents. (The “.” sign in the I_{OH} value means the pin is sourcing that current to the external load, as shown in Figure 4.) To ensure the V_{OH} level listed in the data sheet, the resis-

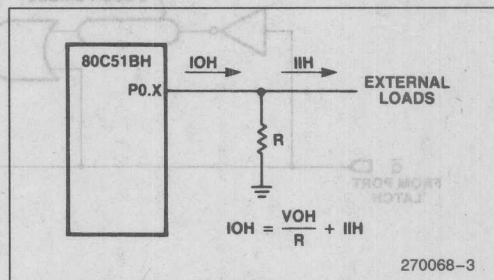


Figure 4a. Conditions defining the minimum value for R. P0.X is emitting a 1 in a bus operation. R must be large enough to not cause I_{OH} to exceed data sheet specifications.

int 1

If a pullup resistor is to be used on a Port 0 pin, its value must satisfy where I_{IH} is the input high current to the external loads.

When the pin goes into a high impedance state, the pulldown resistor will have to sink leakage current from the pin, plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 4b. The Port 0 leakage current is I_{L1} on the data sheet. The resistor should be selected so that the voltage developed across it by these currents will be seen as a logic low by whatever circuits are connected to it (including the 80C51BH). In CMOS/CHMOS applications, 50k Ω is normally a reasonable maximum value.

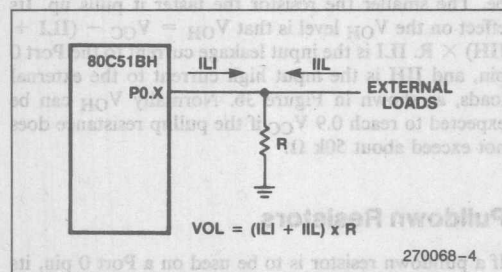


Figure 4b. Conditions defining the maximum value for R. P0.X is in a high impedance state. R must be small enough to keep VOL acceptably low.

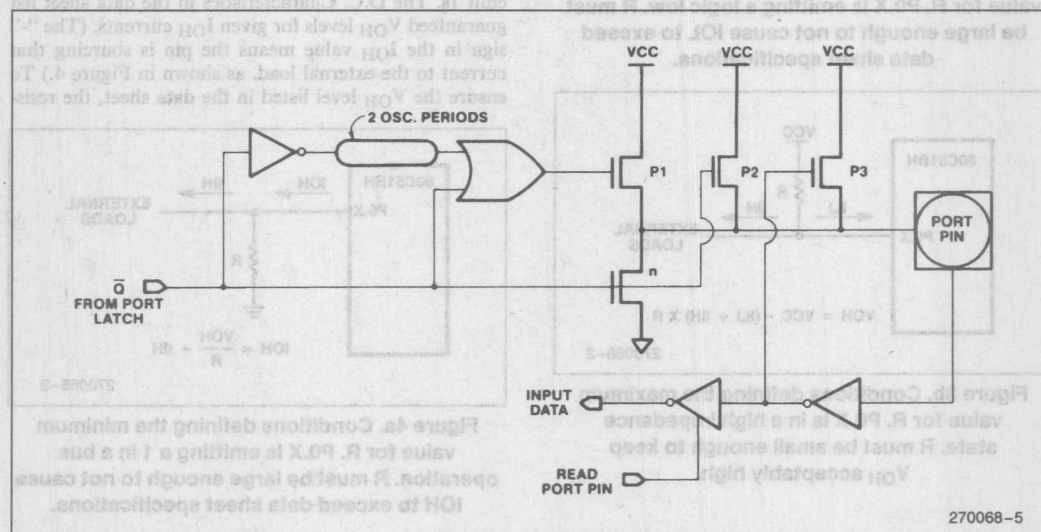


Figure 5. 80C51BH Output Drivers for Ports 1, 2 and 3

DRIVE CAPABILITY OF THE INTERNAL PULLUPS

There's an important difference between HMOS and CHMOS port drivers. The pins of Ports 1, 2, and 3 of the CHMOS parts each have three pullups: strong, normal, and weak, as shown in Figure 5. The strong pullup (p1) is only used during 0-to-1 transitions, to hasten the transition. The weak pullup (p2) is on whenever the bit latch contains a 1. The "normal" pullup (p3) is controlled by the pin voltage itself.

The reason that p3 is controlled by the pin voltage is that if the pin is being used as an input, and the external source pulls it to a low, then turning off p3 makes for a lower I_{IL} . The data sheet shows an " I_{TL} " specification. This is the current that p3 will source during the time the pin voltage is making its 1-to-0 transition. This is what I_{IL} would be if an input low at the pin didn't turn p3 off.

Note, however, that this p3 turn-off mechanism puts a restriction on the drive capacity of the pin if it's being used as an output. If you're trying to output a logic high, and the external load pulls the pin voltage below the pin's V_{IHMIN} spec, p3 might turn off, leaving only the weak p2 to provide drive to the load. To prevent this happening, you need to ensure that the load doesn't draw more than the I_{OH} spec for a valid V_{OH} . The idea is to make sure the pin voltage never falls below its own V_{IHMIN} specification.

POWER CONSUMPTION

The main reason for going to CMOS, of course, is to conserve power. (There are other reasons, but this is the main one.) Conserving power doesn't mean just reducing your electric bill. Nor does it necessarily relate to battery operation, although battery operation without CMOS is pretty unhandy. The main reason for conserving power is to be able to put more functionality into a smaller space. The reduced power consumption allows the use of smaller and lighter power supplies, and less heat being generated allows denser packaging of circuit components. Expensive fans and blowers can usually be eliminated.

A cooler running chip is also more reliable, since most random and wearout failures relate to die temperature. And finally, the lower power dissipation will allow more functions to be integrated onto the chip.

The reason CMOS consumes less power than NMOS is that when it's in a stable state there is no path of conduction from V_{CC} to V_{SS} except through various leakage paths. CMOS does draw current when it's changing states. How much current it draws depends on how often and how quickly it changes states.

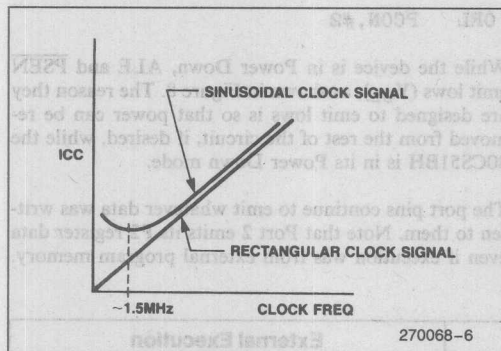


Figure 6. 80C51BH ICC vs. Clock Frequency

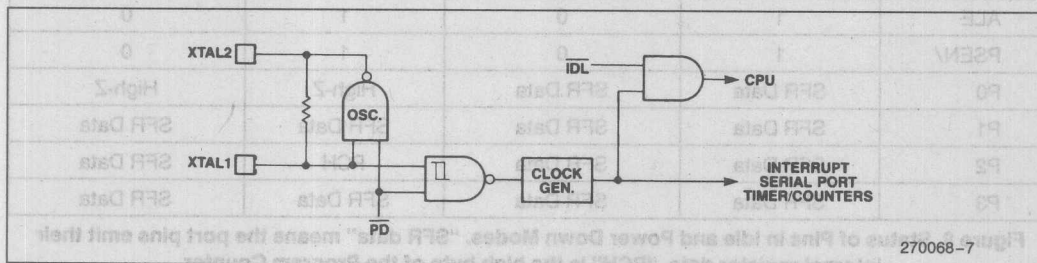


Figure 7. Oscillator and Clock Circuitry Showing Idle and Power Down Hardware

CMOS circuits draw current in sharp spikes during logical transitions. These current spikes are made up of two components. One is the current that flows during the transition time when pullup and pulldown FETs are both active. The average (DC) value of this component is larger when the transition times of the input signals are longer. For this reason, if the current draw is a critical factor in the design, slow rise and fall times should be avoided, even when the system speed doesn't seem to justify a need for nanosecond switching speeds.

The other component is the current that charges stray and load capacitance at the nodes of a CMOS logic gate. The average value of this current spike is its area (integral over time) multiplied by its rep rate. Its area is the amount of charge it takes to raise the node capacitance, C , to V_{CC} . That amount of charge is just $C \times V_{CC}$. So the average value of the current spike is $C \times V_{CC} \times f$, where f is the clock frequency.

This component of current increases linearly with clock frequency. For minimal current draw, the 80C52BH-2 is spec'd to run at frequencies as low as 500 kHz.

Keep in mind, though, that other component of current that is due to slow rise and fall times. A sinusoid is not the optimal waveform to drive the XTAL1 pin with. Yet crystal oscillators, including the one on the 80C51BH, generate sinusoidal waveforms. Therefore, if the on-chip oscillator is being used, you can expect the device to draw more current at 500 kHz, than it does at 1.5 MHz, as shown in Figure 6. If you derive a good sharp square wave from an external oscillator, and use that to drive XTAL1, then the microcontroller will draw less current. But the external oscillator will probably make up the difference.

The 80C51BH has two power-saving features not available in the HMOS devices. These are the Idle and Power Down modes of operation. The on-chip hardware that implements these reduced power modes is shown in Figure 7. Both modes are invoked by software.

Idle: In the Idle Mode ($\overline{IDL} = 0$ in Figure 7), the CPU puts itself to sleep by gating off its own clock. It doesn't stop the oscillator. It just stops the internal clock signal from getting to the CPU. Since the CPU draws 80 to 90 percent of the chip's power, shutting it off represents a fairly significant power savings. The on-chip peripherals (timers, serial port, interrupts, etc.) and RAM continue to function as normal. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle.

The Idle Mode is invoked by setting bit 0 (IDL) of the PCON register. PCON is not bit-addressable, so the bit has to be set by a byte operation, such as

```
ORL PCON,#1
```

The PCON register also contains flag bits GF0 and GF1, which can be used for any general purposes, or to give an indication if an interrupt occurred during normal operation or during Idle. In this application, the instruction that invokes Idle also sets one or both of the flag bits. Their status can then be checked in the interrupt routines.

While the device is in the Idle Mode, ALE and \overline{PSEN} emit logic high (V_{OH}), as shown in Figure 8. This is so external EPROM can be deselected and have its output disabled.

The port pins hold the logical states they had at the time the Idle was activated. If the device was executing out of external program memory, Port 0 is left in a high impedance state and Port 2 continues to emit the high byte of the program counter (using the strong pullups to emit 1s). If the device was executing out of internal program memory, Ports 0 and 2 continue to emit whatever is in the P0 and P2 registers.

There are two ways to terminate Idle. Activation of any enabled interrupt will cause the hardware to clear bit 0 of the PCON register, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that invoked Idle.

The other way is with a hardware reset. Since the clock oscillator is still running, RST only needs to be held active for two machine cycles (24 oscillator periods) to complete the reset. Note that this exit from Idle writes 1s to all the ports, initializes all SFRs to their reset values, and restarts program execution from location 0.

Power Down: In the Power Down Mode ($\overline{PD} = 0$ in Figure 7), the CPU puts the whole chip to sleep by turning off the oscillator. In case it was running from an external oscillator, it also gates off the path to the internal phase generators, so no internal clock is generated even if the external oscillator is still running. The on-chip RAM, however, saves its data, as long as V_{CC} is maintained. In this mode the only I_{CC} that flows is leakage, which is normally in the micro-amp range.

The Power Down Mode is invoked by setting bit 1 in the PCON register, using a byte instruction such as

```
ORL PCON,#2
```

While the device is in Power Down, ALE and \overline{PSEN} emit lows (V_{OL}), as shown in Figure 8. The reason they are designed to emit lows is so that power can be removed from the rest of the circuit, if desired, while the 80CS51BH is in its Power Down mode.

The port pins continue to emit whatever data was written to them. Note that Port 2 emits its P2 register data even if execution was from external program memory.

Pin	Internal Execution		External Execution	
	Idle	Power Down	Idle	Power Down
ALE	1	0	1	0
$\overline{PSEN}/$	1	0	1	0
P0	SFR Data	SFR Data	High-Z	High-Z
P1	SFR Data	SFR Data	SFR Data	SFR Data
P2	SFR Data	SFR Data	PCH	SFR Data
P3	SFR Data	SFR Data	SFR Data	SFR Data

Figure 8. Status of Pins in Idle and Power Down Modes. "SFR data" means the port pins emit their internal register data. "PCH" is the high byte of the Program Counter.

Port 0 also emits its P0 register data, but if execution was from external program memory, the P0 register data is FF. The oscillator is stopped, and the part remains in this state as long as V_{CC} is held, and until it receives an external reset signal.

The only exit from Power Down is a hardware reset. Since the oscillator was stopped, RST must be held active long enough for the oscillator to re-start and stabilize. Then the reset function initializes all the Special Function Registers (ports, timers, etc.) to their reset values, and re-starts the program from location 0. Therefore, timer reloads, interrupt enables, baud rates, port status, etc. need to be re-established. Reset does not affect the content of the on-chip data RAM. If V_{CC} was held during Power Down, the RAM data is still good.

USING THE POWER DOWN MODE

The software-invoked Power Down feature offers a means of reducing the power consumption to a mere trickle in systems which are to remain dormant for some period of time, while retaining important data.

The user should give some thought to what state the port pins should be left in during the time the clock is stopped, and write those values to the port latches before invoking Power Down.

If V_{CC} is going to be held to the entire circuit, one would want to write values to the port latches that would deselect peripherals before invoking Power Down. For example, if external memory is being used, the P2 SFR should be loaded with a value which will not generate an active chip select to any memory device.

In some applications, V_{CC} to part of the system may be shut off during Power Down, so that even quiescent and standby currents are eliminated. Signal lines that connect to those chips must be brought to a logic low, whether the chip in question is CMOS, NMOS, or TTL, before V_{CC} is shut off to them. CMOS pins have parasitic pn junctions to V_{CC} , which will be forward biased if V_{CC} is reduced to zero while the pin is held at a logic high. NMOS pins often have FETs that look like diodes to V_{CC} . TTL circuits may actually be damaged by an input high if $V_{CC} = 0$. That's why the 80C51BH outputs lows at ALE and \overline{PSEN} during Power Down.

Figure 9 shows a circuit that can be used to turn V_{CC} off to part of the system during Power Down. The circuit will ensure that the secondary circuit is not de-energized until after the 80C31BH is in Power Down, and that the 80C31BH does not receive a reset (terminating the Power Down mode) before the secondary circuit is re-energized. Therefore, the program memory itself can be part of the secondary circuit.

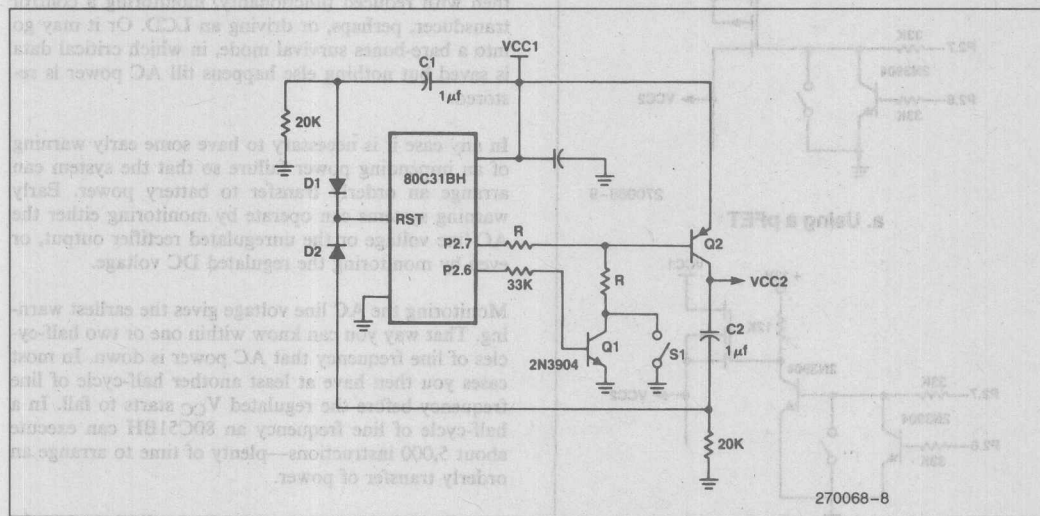


Figure 9. The 80C31BH de-energizes part of the circuit (V_{CC2}) when it goes into Power Down. Selections of R and Q2 depend on V_{CC2} current draw.

In Figure 9, when V_{CC} is switched on to the 80C31BH, capacitor C1 provides a power-on reset. The reset function writes 1s to all the port pins. The 1 at P2.6 turns Q1 on, enabling V_{CC} to the secondary circuit through transistor Q2. As the 80C31BH comes out of reset, Port 2 commences emitting the high byte of the Program Counter, which results in the P2.7 and P2.6 pins outputting 0s. The 0 at P2.7 ensures continuation of V_{CC} to the secondary circuit.

The system software must now write a 1 to P2.7 and a 0 to P2.6 in the Port 2 SFR, P2. These values will not appear at the Port 2 pins as long as the device is fetching instructions from external program memory. However, whenever the 80C31BH goes into Power Down, these values will appear at the port pins, and will shut off both transistors, disabling V_{CC} to the secondary circuit.

Closing the switch S1 re-energizes the secondary circuit, and at the same time sends a reset through C2 to the 80C31BH to wake it up. The diode D1 is to prevent C1 from hogging current from C2 during this secondary reset. D2 prevents C2 from discharging through the RST pin when V_{CC} to the secondary circuit goes to zero.

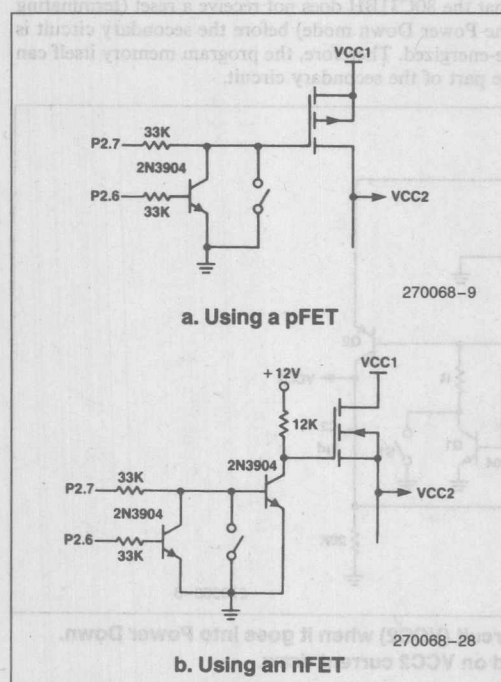


Figure 10. Using Power MOSFETs to Control V_{CC2}

USING POWER MOSFETs to CONTROL V_{CC}

Power MOSFETs are gaining in popularity (and availability). The easiest way to control V_{CC} is with a Logic Level pFET, as shown in Figure 10a. This circuit allows the full V_{CC} to be used to turn the device on. Unfortunately, power pFETs are not economically competitive with bipolar transistors of comparable ratings.

Power nFETs are both economical and available, and can be used in this application if a DC supply of higher voltage is available to drive the gate. Figure 10b shows how to implement a V_{CC} switch using a power nFET and a (nominally) +12V supply. The problem here is that if the device is on, its source voltage is +5V. To maintain the on state, the gate has to be another 5 or 10V above that. The "12V" supply is not particularly critical. A minimally filtered, unregulated rectifier will suffice.

BATTERY BACKUP SYSTEMS

Here we consider circuits that normally draw power from the AC line, but switch to battery operation in the event of a power failure. We assume that in battery operation high-current loads will be allowed to die along with the AC power. The system may continue then with reduced functionality, monitoring a control transducer, perhaps, or driving an LCD. Or it may go into a bare-bones survival mode, in which critical data is saved but nothing else happens till AC power is restored.

In any case it is necessary to have some early warning of an impending power failure so that the system can arrange an orderly transfer to battery power. Early warning systems can operate by monitoring either the AC line voltage or the unregulated rectifier output, or even by monitoring the regulated DC voltage.

Monitoring the AC line voltage gives the earliest warning. That way you can know within one or two half-cycles of line frequency that AC power is down. In most cases you then have at least another half-cycle of line frequency before the regulated V_{CC} starts to fall. In a half-cycle of line frequency an 80C51BH can execute about 5,000 instructions—plenty of time to arrange an orderly transfer of power.

The circuit in Figure 11 uses a Zener diode to test the line voltage each half-cycle, and a junction transistor to pass the information on to the 80C51BH. (Obviously a voltage comparator with a suitable reference source can

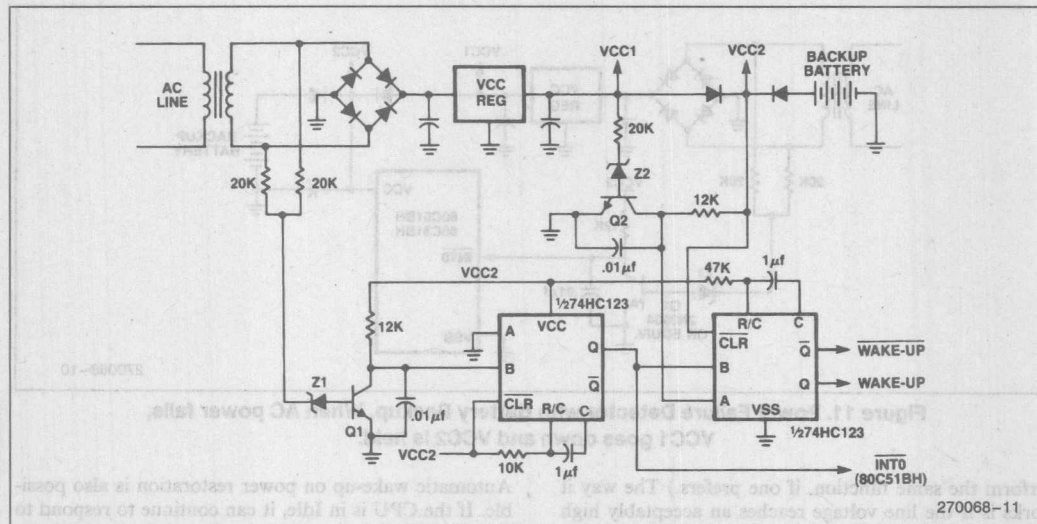


Figure 12. Power Failure Detector uses retriggerable one-shots to flag impending power outage and generate automatic wake-up when power returns.

Having been awakened, the 80C51BH will stay awake for at least another half-cycle of line frequency (another 5,000 or so instructions) before possibly being told to arrange another transfer of power. Consequently, if the line voltage is jittering erratically around the switchover point (determined by diode Z1), the system will limp along executing in half-cycle units of line frequency.

On the other hand, if the power outage is real and lengthy, VCC1 will eventually fall below the level at which the backup battery takes over. The backup battery maintains power to the 80C51BH, and to the 74HC123, and to whatever other circuits are being protected during this outage. The battery voltage must be high enough to maintain VCCMIN specs to the 80C51BH.

If the microcontroller is an 80C31BH, executing out of external ROM, and if the C31BH is put into Idle during the power outage, then the external ROM must also be supplied by the battery. On the other hand, if the C31BH is put into Power Down during the outage, then the ROM can be allowed to die with the AC power. The considerations here are the same as in Figure 9: VCC to the ROM is still up at the time Power Down is invoked, and we must ensure (through selection of diode Z2 in Figure 12) that the 80C31BH is not awakened till ROM power is back in spec.

POWER SWITCHOVER CIRCUITS

Battery backup systems need to have a way for the protected circuits to draw power from the line-operated power supply when that source is available, and to switch over to battery power when required. The switchover circuit is simple if the entire system is to be battery powered in the event of a line power outage. In that case a pair of diodes suffice, as shown in Figure 12, provided VCCMIN specs are still met after the diode drop has been subtracted from its respective power source.

The situation becomes more complicated when part of the circuit is going to be allowed to die when the AC power goes out. In that case it is difficult to maintain equal VCCs to protected and unprotected circuits (and possibly dangerous not to).

The problem can be alleviated by using a Schottky diode instead of a 1N4001, for its lower forward voltage drop. The 1N5820, for example, has a forward drop of about 0.35V at 1A.

Other solutions are to use a transistor or power MOSFET switch, as shown in Figure 13. With minor modifications this switch can be controlled by port pins.

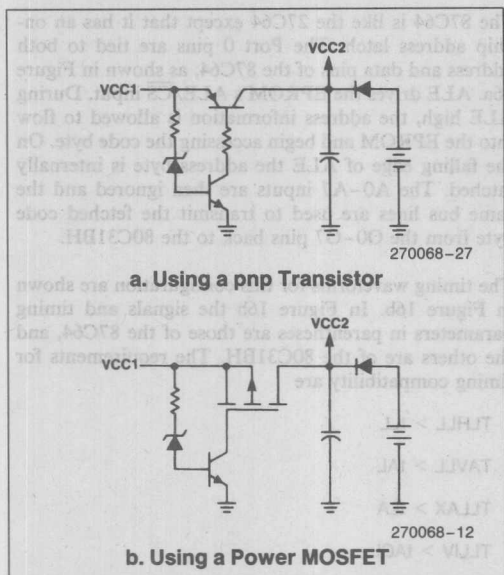


Figure 13. Power Switchover Ckts.

80C31BH + CHMOS EPROM

The 27C64 and 87C64 are Intel's 8K byte CHMOS EPROMs. The 27C64 requires an external address latch, and can be used with the 80C31BH as shown in Figure 14a. In most 8031 + 2764 (HMOS) appli-

cations, the 2764's Chip Enable (\overline{CE}) pin is hardwired to ground (since it's normally the only program memory on the bus). This can be done with the CHMOS versions as well, but there is some advantage in connecting \overline{CE} to ALE, as shown in Figure 14a. The advantage is that if the 80C31BH is put into Idle mode, since ALE goes to a 1 in that mode, the 27C64 will be deselected and go into a low current standby mode.

The timing waveforms for this configuration are shown in Figure 14b. In Figure 14b the signals and timing parameters in parenthesis are those of the 27C64, and the others are of the 80C31BH, except Tprop is a parameter of the address latch. The requirements for timing compatibility are

- TAVIV - Tprop > tACC
- TLLIV > tCE
- TPLIV > tOE
- TPXIZ > tDF

If the application is going to use the Power Down mode then we have another consideration: In Idle, $ALE = \overline{PSEN} = 1$, and in Power Down, $ALE = \overline{PSEN} = 0$. In a realistic application there are likely to be more chips in the circuit than are shown in Figure 14, and it is likely that the nonessential ones will have their VCC removed while the CPU is in Power Down. In that case the EPROM and the address latch should be among the chips that have VCC removed, and logic lows are exactly what are required at ALE and PSEN.

But if VCC is going to be maintained to the EPROM during Power Down, then it will be necessary to de-

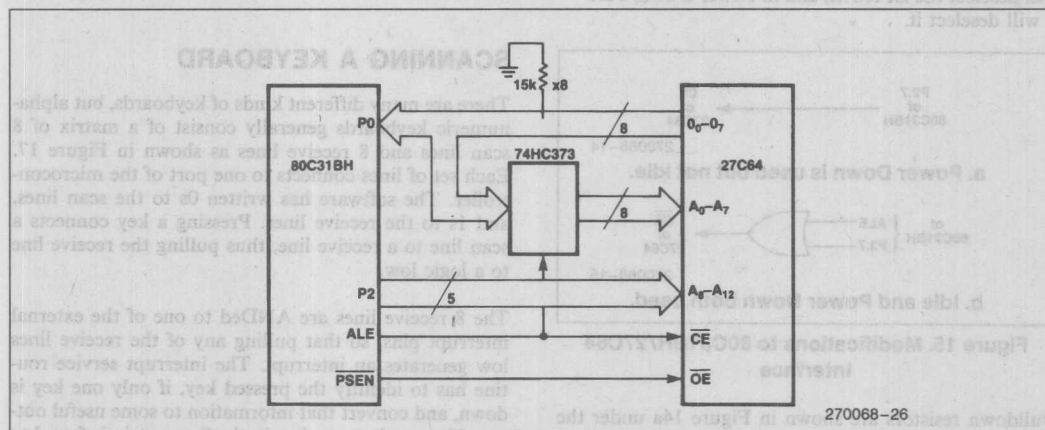


Figure 14a. 80C31BH + 27C64

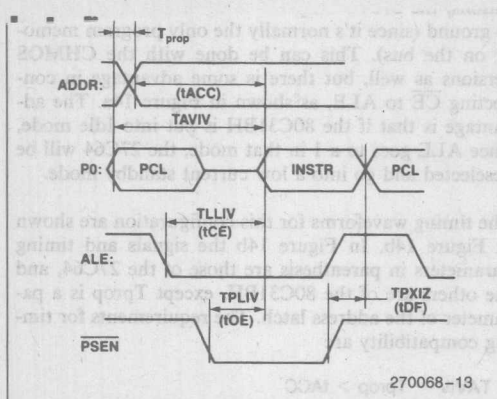


Figure 14b. Timing Waveforms for 80C31BH + 27C64

select the EPROM when the CPU is in Power Down. If Idle is never invoked, \overline{CE} of the EPROM can be connected to P2.7 of the 80C31BH, as shown in Figure 15a. In normal operation, P2.7 will be emitting the MSB of the Program Counter, which is 0 if the program contains less than 32K of code. Then when the CPU goes into Power Down, the Port 2 pins emit P2 SFR data, which puts a 1 at P2.7, thus deselecting the EPROM.

If Idle and Power Down are both going to be used, \overline{CE} of the EPROM can be driven by the logical OR of ALE and P2.7, as shown in Figure 15b. In Idle, ALE = 1 will deselect the EPROM, and in Power Down, P2.7 = 1 will deselect it.

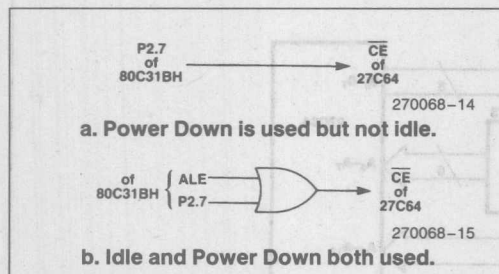


Figure 15. Modifications to 80C31BH/27C64 Interface

Pulldown resistors are shown in Figure 14a under the assumption that something on the bus is going to have its V_{CC} removed during Power Down. If this is not the case, pullups can be used as well as pulldowns.

The 87C64 is like the 27C64 except that it has an on-chip address latch. The Port 0 pins are tied to both address and data pins of the 87C64, as shown in Figure 16a. ALE drives the EPROM's ALE/ \overline{CS} input. During ALE high, the address information is allowed to flow into the EPROM and begin accessing the code byte. On the falling edge of ALE the address byte is internally latched. The A0-A7 inputs are then ignored and the same bus lines are used to transmit the fetched code byte from the 00-07 pins back to the 80C31BH.

The timing waveforms for this configuration are shown in Figure 16b. In Figure 16b the signals and timing parameters in parentheses are those of the 87C64, and the others are of the 80C31BH. The requirements for timing compatibility are

TLHLL > tLL

TAVLL > tAL

TLLAX > tLA

TLLIV > tACL

TPLIV > tOE

TLLPL > tCOE

TPXIZ > tOHZ

The same considerations apply to the 87C64 as to the 27C64 with regards to the Idle and Power Down modes. Basically you want $\overline{CS} = 1$ if V_{CC} is maintained to the EPROM, and $\overline{CS} = \overline{OE} = 0$ if V_{CC} is removed.

SCANNING A KEYBOARD

There are many different kinds of keyboards, but alphanumeric keyboards generally consist of a matrix of 8 scan lines and 8 receive lines as shown in Figure 17. Each set of lines connects to one port of the microcontroller. The software has written 0s to the scan lines, and 1s to the receive lines. Pressing a key connects a scan line to a receive line, thus pulling the receive line to a logic low.

The 8 receive lines are ANDed to one of the external interrupt pins, so that pulling any of the receive lines low generates an interrupt. The interrupt service routine has to identify the pressed key, if only one key is down, and convert that information to some useful output. If more than one key in the line matrix is found to be pressed, no action is taken. (This is a "two key lock-out" scheme.)

On some keyboards, certain keys (Shift, Control, Escape, etc.) are not a part of the line matrix. These keys would connect directly to a port pin on the microcontroller, and would not cause lock-out if pressed simultaneously with a matrix key, nor generate an interrupt if pressed singly.

Normally the microcontroller would be in idle mode when a key has not been pressed, and another task is not in progress. Pressing a matrix key generates an in-

terrupt, which terminates the Idle. The interrupt service routine would first call a 30 ms (or so) delay to debounce the key, and then set about the task of identifying which key is down.

First, the current state of the receive lines is latched into an internal register. If a single key is down, all but one of the lines would be read as 1s. Then 0s are written to the receive lines and 1s to the scan lines, and the scan lines are read. If a single key is down, all but one of

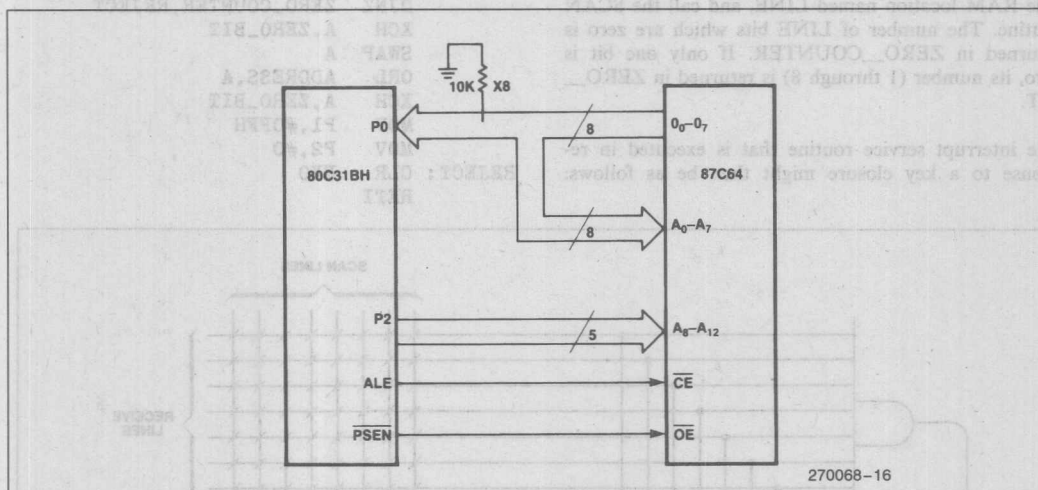


Figure 16a. 80C31BH + 87C64

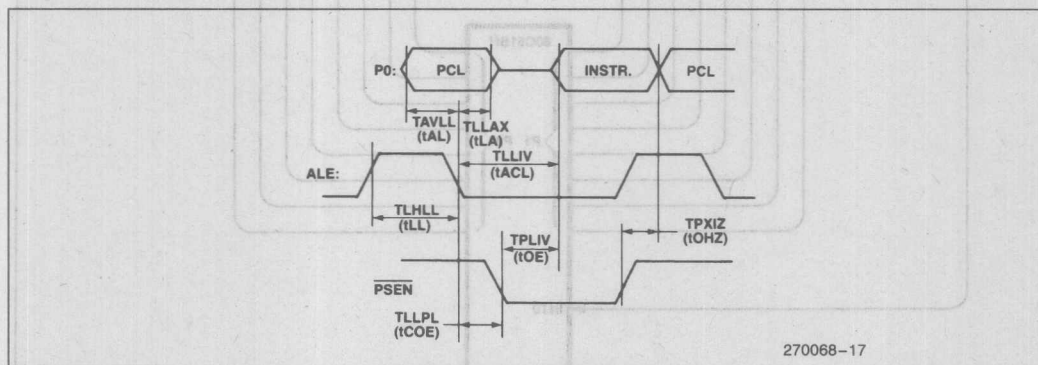


Figure 16b. Timing Waveforms for 80C31BH + 87C64

these lines would be read as 1s. By locating the single 0 in each set of lines, the pressed key can be identified. If more than one matrix key is down, one or both sets of lines will contain multiple 0s.

A subroutine is used to determine which of 8 bits in either set of lines is 0, and whether more than one bit is 0. Figure 18 shows a subroutine (SCAN) which does that using the 8051's bit-addressing capability. To use the subroutine, move the line data into a bit-addressable RAM location named LINE, and call the SCAN routine. The number of LINE bits which are zero is returned in ZERO_COUNTER. If only one bit is zero, its number (1 through 8) is returned in ZERO_BIT.

The interrupt service routine that is executed in response to a key closure might then be as follows:

RESPONSE_TO_KEY_CLOSURE:

```
CALL DEBOUNCE_DELAY
MOV LINE,P1; ;See Figure 17.
CALL SCAN
DJNZ ZERO_COUNTER,REJECT
MOV ADDRESS,ZERO_BIT
MOV P2,#OFFH; ;See Figure 17.
MOV P1,#0
MOV LINE,P2
CALL SCAN
DJNZ ZERO_COUNTER,REJECT
XCH A,ZERO_BIT
SWAP A
ORL ADDRESS,A
XCH A,ZERO_BIT
MOV P1,#OFFH
MOV P2,#0
REJECT: CLR EX0
        RETI
```

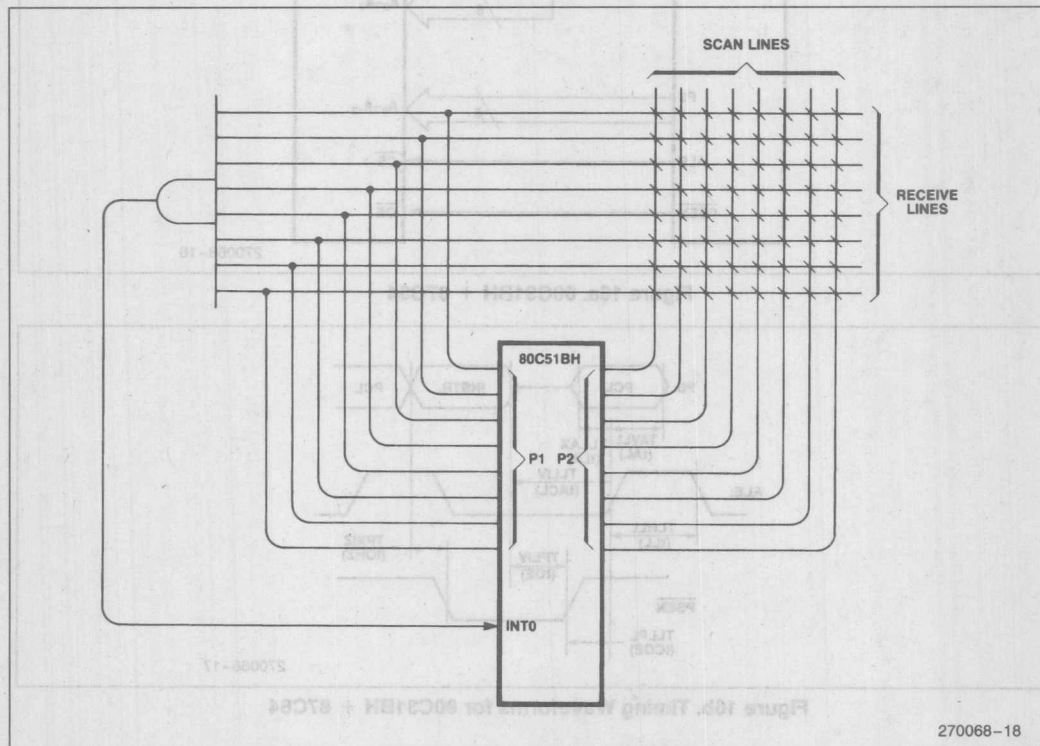


Figure 17. Scanning a Keyboard

```

SCAN:  MOV     ZERO_COUNTER,#0 ; ZERO_COUNTER counts the number of 0s in LINE.
        JB     LINE_0.ONE      ; Test LINE bit 0.
        INC     ZERO_COUNTER   ; If LINE_0 = 0, increment ZERO_COUNTER
        MOV     ZERO_BIT,#1    ; and record that line number 1 is active.
ONE:    JB     LINE_1.TWO      ; Procedure continues for other LINE bits.
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#2    ; Line number 2 is active.
TWO:    JB     LINE_2.THREE
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#3    ; Line number 3 is active.
THREE:  JB     LINE_3.FOUR
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#4    ; Line number 4 is active.
FOUR:   JB     LINE_4.FIVE
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#5    ; Line number 5 is active.
FIVE:   JB     LINE_5.SIX
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#6    ; Line number 6 is active.
SIX:    JB     LINE_6.SEVEN
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#7    ; Line number 7 is active.
SEVEN:  JB     LINE_7.EIGHT
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#8    ; Line number 8 is active.
EIGHT:  RET

```

270068-19

Figure 18. Subroutine SCAN Determines Which of 8 Bits in LINE is Zero

Notice that `RESPONSE_TO_KEY_CLOSURE` does not change the Accumulator, the PSW, nor any of the registers R0 through R7. Neither do `SCAN` or `DEBOUNCE_DELAY`.

What we come out with then is a one-byte key address (`ADDRESS`) which identifies the pressed key. The key's scan line number is in the upper nibble of `ADDRESS`, and its receive line number is in the lower nibble. `ADDRESS` can be used in a look-up table to generate a key code to transmit to a host computer, and/or to a display device.

The keyboard interrupt itself must be edge-triggered, rather than level-activated, so that the interrupt routine is invoked when a key is pressed, and is not constantly being repeated as long as the key is held down. In edge-triggered mode, the on-chip hardware clears the interrupt flag (`EX0`, in this case) as the service routine is being vectored to. In this application, however, contact bounce will cause several more edges to occur after the service routine has been vectored to, during the `DEBOUNCE_DELAY` routine. Consequently it is necessary to clear `EX0` again in software before executing `RETI`.

The debounce delay routine also takes advantage of the Idle mode. In this routine a timer must be preloaded with a value appropriate to the desired length of delay. This would be

$$\text{timer preload} = \frac{(\text{osc kHz}) \times (\text{delay time ms})}{12}$$

For example, with a 3.58 MHz oscillator frequency, a 30 ms delay could be obtained using a preload value of -8950, or DD0A, in hex digits.

In the debounce delay routine (Figure 19), the timer interrupt is enabled and set to a higher priority than the keyboard interrupt, because as we invoke Idle, the keyboard interrupt is still "in progress". An interrupt of the same priority will not be acknowledged, and will not terminate the Idle mode. With the timer interrupt set to priority 1, while the keyboard interrupt is a priority 0, the timer interrupt, when it occurs, will be acknowledged and will wake up the CPU. The timer interrupt service routine does not itself have to do anything. The service routine might be nothing more than a single `RETI` instruction. `RETI` from the timer interrupt service routine then returns execution to the debounce delay routine, which shuts down the timer and returns execution to the keyboard service routine.

DRIVING AN LCD

An LCD (Liquid Crystal Display) consists of a backplane and any number of segments or dots which will be used to form the image being displayed. Applying a voltage (nominally 4 or 5V) between any segment and the backplane causes the segment to darken. The only catch is that the polarity of the applied voltage has to be periodically reversed, or else a chemical reac-

```

DEBOUNCE_DELAY:
MOV     TL1,#TL1_PRELOAD ; Preload low byte.
MOV     TH1,#TH1_PRELOAD ; Preload high byte.
SETB    ET1               ; Enable Timer 1 interrupt.
SETB    PT1               ; Set Timer 1 interrupt to high priority.
SETB    TR1               ; Start timer running.
ORL     PCON,#1           ; Invoke Idle mode.
; The next instruction will not be executed until the delay times out.

CLR     TR1               ; Stop the timer.
CLR     PT1               ; Back to priority 0 (if desired).
CLR     ET1               ; Disable Timer 1 interrupt (if desired).
RET                                ; Continue keyboard scan.

```

270068-20

Figure 19. Subroutine DEBOUNCE_DELAY Puts the 80C51BH into Idle During the Delay Time

tion takes place in the LCD which causes deterioration and eventual failure of the liquid crystal.

To prevent this happening, the backplane and all the segments are driven with an AC signal, which is derived from a rectangular voltage waveform. If a segment is to be "off" it is driven by the same waveform as the backplane. Thus it is always at backplane potential. If the segment is to be "on" it is driven with a waveform that is the inverse of the backplane waveform. Thus it has about 5V of periodically changing polarity between it and the backplane.

With a little software overhead, the 80C51BH can perform this task without the need for additional LCD drivers. The only drawback is that each LCD segment uses up one port pin, and the backplane uses one more. If more than, say, two 7-segment digits are being driven, there aren't many port pins left for other tasks. Nevertheless, assuming a given application leaves enough port pins available to support this task, the considerations for driving the LCD are as follows.

Suppose, for example, it is a 2-digit display with a decimal point. One port (TENS_DIGIT) connects to the 7 segments of the tens digit plus the backplane. Another port (ONES_DIGIT) connects to a decimal point plus the 7 segments of the ones digit.

One of the 80C51BH's timers is used to mark off half-periods of the drive voltage waveform. The LCD drive waveform should have a rep rate between 30 and 100 Hz, but it's not very critical. A half-period of 12 ms will set the rep rate to about 42 Hz. The preload/reload value to get 12 ms to rollover is the 2's complement negative of the oscillator frequency in kHz: if the oscillator frequency is 3.58 MHz, the reload value is -3580, or F204 in hex digits.

Now, the 80C51BH would normally be in Idle, to conserve power, during the time that the LCD and other

tasks are not requiring servicing. When the timer rolls over it generates an interrupt, which brings the 80C51BH out of Idle. The service routine reloads the timer (for the next rollover), and inverts the logic levels of all the pins that are connected to the LCD. It might look like this:

```

LCD_DRIVE_INTERRUPT:
MOV     TL1,#LOW( - XTAL_FREQ)
MOV     TH1,#HIGH( - XTAL_FREQ)
XRL     TENS_DIGIT,#OFFH
XRL     ONES_DIGIT,#OFFH
RETI

```

To update the display, one would use a look-up table to generate the characters. In the table, "on" segments are represented as 1s, and "off" segments as 0s. The backplane bit is represented as a 0. The quantity to be displayed is stored in RAM as a BCD value. The look-up table operates on the low nibble of the BCD value, and produces the bit pattern that is to be written to either the ones digit or the tens digit. Before the new patterns can be written to the LCD, the LCD drive interrupt has to be disabled. That is to prevent a polarity reversal from taking place between the times the two digits are written. An update subroutine is shown in Figure 20.

USING AN LCD DRIVER

As was noted, driving an LCD directly with an 80C51BH uses a lot of port pins. LCD drivers are available in CMOS to interface an 80C51BH to a 4-digit display using only 7 of the C51BH's I/O pins. Basically, the C51BH tells the LCD driver what digit is to be displayed (4 bits) and what position it is to be displayed in (2 bits), and toggles a Chip Select pin to tell the driver to latch this information. The LCD driver generates the display characters (hex digits), and takes care of the polarity reversals using its own RC oscillator to generate the timing.

Figure 25 shows an 80C51BH working with an ICM7211M to drive a 4-digit LCD, and the software that updates the display.

One could equally well send information to the LCD driver over the bus. In that case, one would set up the Accumulator with the digit select and data input bits, and execute a MOVX@ R0,A instruction. The LCD driver's chip select would be driven by the CPU's WR signal. This is a little easier in software than the direct bit manipulation shown in Figure 21. However, it uses more I/O pins, unless there is already some external memory involved. In that case, no extra pins are used up by adding the LCD driver to the bus.

RESONANT TRANSDUCERS

Analog transducers are often used to convert the value of a physical property, such as temperature, pressure, etc., to an analog voltage. These kinds of transducers then require an analog-to-digital converter to put the measurement into a form that is compatible with a digital control system. Another kind of transducer is now becoming available that encodes the value of the physical property into a signal that can be directly read by a digital control system. These devices are called resonant transducers.

Resonant transducers are oscillators whose frequency depends in a known way on the physical property being measured. These devices output a train of rectangular pulses whose repetition rate encodes the value of the quantity being measured. The pulses can in most cases be fed directly into the 80C51BH, which then measures either the frequency or period of the incoming signal, basing the measurement on the accuracy of its own clock oscillator. The 80C51BH can even do this in its sleep; that is, in Idle.

When the frequency or period measurement is completed, the C51BH wakes itself up for a very short time to perform a sanity check on the measurement and convert it in software to any scaling of the measured quantity that may be desired. The software conversion can include corrections for nonlinearities in the transducer's transfer function.

Resolution is also controlled by software, and can even be dynamically varied to meet changing needs as a situation becomes more critical. For example, in a process controller you can increase your resolution ("fine tune" the control, as it were) as the process approaches its target.

The nominal reference frequency of the output signal from these devices is in the range of 20 Hz to 500 kHz, depending on the design. Transducers are available that have a full scale frequency shift 2 to 1. The transducer operates from a supply voltage range of 3V to 20V, which means it can operate from the same supply voltage as the 80C51BH. At 5V, the transducer draws less than 5 mA (Reference 7). It can normally be connected directly to one of the C51BH's port pins, as shown in Figure 22.

FREQUENCY MEASUREMENTS

Measuring a frequency means counting pulses for a known sample time. Two timer/counters can be used, one to mark off the sample time and one to count pulses. If the frequency being counted doesn't exceed 50 kHz or so, one may equally well connect the transducer signal to one of the external interrupt pins, and count pulses in software. That frees up one timer, with very little cost in CPU time.

The count that is directly obtained is TxF, where T is the sample time and F is the frequency. The full scale

UPDATE_LCD:		
CLR	ET1	; Disable LCD drive interrupt.
MOV	DPTR,#TABLE_ADDRESS	; Look-up table begins at TABLE_ADDRESS
MOV	A,BCD_VALUE	; Digits to be displayed.
SWAP	A	; Move tens digit to low nibble.
ANL	A,#0FH	; Mask off high nibble.
MOVC	A,@A+DPTR	; Tens digit pattern to accumulator.
MOV	TENS_DIGIT,A	; Update LCD tens digit.
MOV	A,BCD_VALUE	; Digits to be displayed.
ANL	A,#0FH	; Mask off tens digit.
MOVC	A,@A+DPTR	; Ones digit pattern to accumulator.
MOV	C,DECIMAL_POINT	; Add decimal point to segment
MOV	ACC.7,C	; pattern. Update LCD decimal point
MOV	ONES_DIGIT,A	; and ones digit.
SETB	ET1	; Re-enable LCD drive interrupt.
RET		

270068-21

Figure 20. UPDATE_LCD Routine Writes Two Digits to an LCD

range is $T_x(F_{max}-F_{min})$. For n-bit resolution

$$1 \text{ LSB} = \frac{T_x(F_{max}-F_{min})}{2^n}$$

Therefore the sample time required for n-bit resolution is

$$T = \frac{2^n}{F_{max}-F_{min}}$$

For example, 8-bit resolution in the measurement of a frequency that varies between 7 kHz and 9 kHz would require, according to this formula, a sample time of 128 ms. The maximum acceptable frequency count would be $128 \text{ ms} \times 9 \text{ kHz} = 1152 \text{ counts}$. The minimum would be 896 counts. Subtracting 896 from each frequency count (or presetting the frequency counter to $-896 = 0FC80H$) would allow the frequency to be reported on a scale of 0 to FF in hex digits.

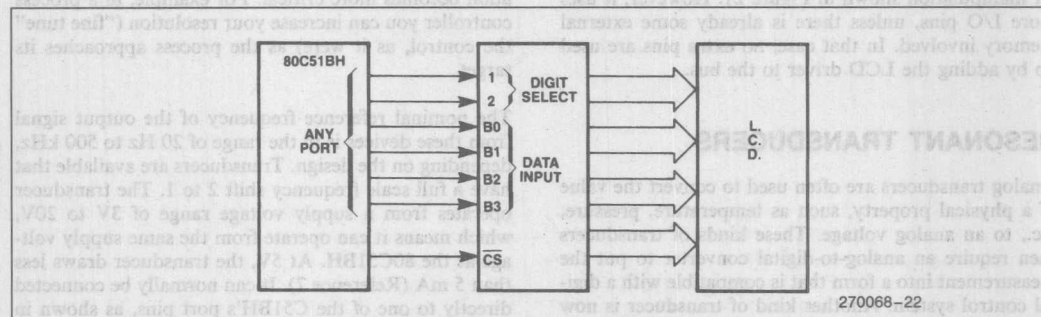


Figure 21a. Using an LCD Driver

```

UPDATE_LCD:
MOV     A, DISPLAY_HI      ; High byte of 4-digit display.
SETB    DIGIT_SELECT_2     ; Select leftmost digit of LCD.
SETB    DIGIT_SELECT_1     ; (Digit address = 11B.)
CALL    SHIFT_AND_LOAD     ; High nibble of high byte to selected digit.
CLR     DIGIT_SELECT_1     ; Select second digit of LCD (address = 10B.)
CALL    SHIFT_AND_LOAD     ; Low nibble of high byte to selected digit.
MOV     A, DISPLAY_LO      ; Low byte of 4-digit display.
CLR     DIGIT_SELECT_2     ; Select third digit of LCD.
SETB    DIGIT_SELECT_1     ; (Digit address = 01B.)
CALL    SHIFT_AND_LOAD     ; High nibble of low byte to selected digit.
CLR     DIGIT_SELECT_1     ; Select fourth digit (address = 00B.)
CALL    SHIFT_AND_LOAD     ; Low nibble of low byte to selected digit.
RET

SHIFT_AND_LOAD:
RLC     A                   ; MSB to carry bit (CY).
MOV     DATA_INPUT_B3,C   ; CY to Data Input pin B3.
RLC     A                   ; Next bit to CY.
MOV     DATA_INPUT_B2,C   ; CY to Data Input pin B2.
RLC     A                   ; Next bit to CY.
MOV     DATA_INPUT_B1,C   ; CY to Data Input pin B1.
RLC     A                   ; Last bit to CY.
MOV     DATA_INPUT_B0,C   ; CY to Data Input pin B0.
CLR     CHIP_SELECT        ; Toggle Chip Select.
SETB    CHIP_SELECT        ; 0-to-1 transition latches info.
RET
    
```

Figure 21b. UPDATE_LCD Routine Writes 4 Digits to an LCD Driver

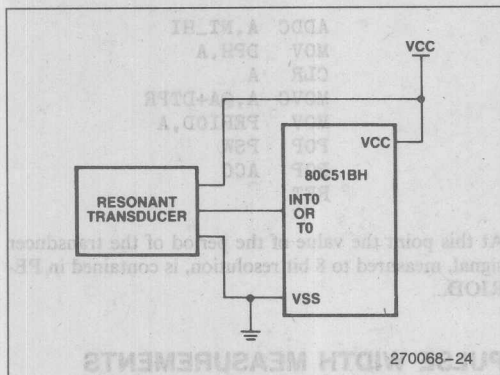


Figure 22. Resonant Transducer Does Not Require an A/D Converter

To implement the measurement, one timer is used to establish the sample time. The timer is preset to a value that causes it to roll over at the end of the sample time, generating an interrupt and waking the CPU from its Idle mode. The required preset value is the 2's complement negative of the sample time measured in machine cycles. The conversion from sample time to machine cycles is to multiply it by 1/12 the clock frequency. For example, if the clock frequency is 12 MHz, then a sample time of 128 ms is

$$(128 \text{ ms}) \times (12000 \text{ kHz})/12 = 128000 \text{ machine cycles.}$$

Then the required preset value to cause the timer to roll over in 128 ms is

$$-128000 = \text{FE0C00, in hex digits.}$$

Note that the preset value is 3 bytes wide whereas the timer is only 2 bytes wide. This means the timer must be augmented in software in the timer interrupt routine to three bytes. The 80C51BH has a DJNZ instruction (decrement and jump if not zero) that makes it easier to code the third timer byte to count down instead of up. If the third timer byte counts down, its reload value is the 2's complement of what it would be for an up-counter. For example, if the 2's complement of the sample time is FE0C00, then the reload value for the third timer byte would be 02, instead of FE. The timer interrupt routine might then be:

```
TIMER_INTERRUPT_ROUTINE:
    DJNZ    THIRD_TIMER_BYTE,OUT
    MOV     TLO,#0
    MOV     TH0,#0CH
    MOV     THIRD_TIMERBYTE,#2
    MOV     FREQUENCY,COUNTER_LO
;Preset COUNTER to -896:
    MOV     COUNTER_LO,#80H
    MOV     COUNTER_HI,#0FCH
OUT:    RETI
```

At this point the value of the frequency of the transducer signal, measured to 8 bit resolution, is contained in FREQUENCY. Note that the timer can be reloaded on the fly. Note too that the timer can be reloaded on the fly. Note too that for 8-bit resolution only the low byte of the frequency counter needs to be read, since the high byte is necessarily 0. However, one may want to test the high byte to ensure that it is zero, as a sanity check on the data. Both bytes, of course must be reloaded.

PERIOD MEASUREMENTS

Measuring the period of the transducer signal means measuring the total elapsed time over a known number, N, of transducer pulses. The quantity that is directly measured is NT, where T is the period of the transducer signal in machine cycles. The relationship between T in machine cycles and the transducer frequency F in arbitrary frequency units is

$$T = \frac{F_{\text{xtal}}}{F} \times (1/12),$$

where Fxtal is the 80C51BH clock frequency, in the same units as F.

The full scale range then is Nx (Tmax-Tmin). For n-bit resolution.

$$1 \text{ LSB} = \frac{N_s(T_{\text{max}}-T_{\text{min}})}{2^n}$$

Therefore the number of periods over which the elapsed time should be measured is

$$N = \frac{2^n}{T_{\text{max}}-T_{\text{min}}}$$

However, N must also be an integer. It is logical to evaluate the above formula (don't forget Tmax and Tmin have to be in machine cycles) and select for N the next higher integer. This selection gives a period measurement that has somewhat more than n-bit resolution, but it can be scaled back if desired.

For example, suppose we want 8-bit resolution in the measurement of the period of a signal whose frequency varies from 7.1 kHz to 9 kHz. If the clock frequency is 12 MHz, then Tmax is (12000 kHz/7.1 kHz) x (1/12) = 141 machine cycles. Tmin is 111 machine cycles. The required value for N, then, is 256/(141-111) = 8.53 periods, according to the formula. Using N = 9 periods will give a maximum NT value of 141 x 9 = 1269 machine cycles. The minimum NT will be 111 x 9 = 999 machine cycles. A lookup table can be used to

scale these values back to a range of 0 to 255, giving precisely the 8-bit resolution desired.

To implement the measurement, one timer is used to measure the elapsed time, NT. The transducer is connected to one of the external interrupt pins, and this interrupt is configured to the transition-activated mode. In the transition-activated mode every 1-to-0 transition in the transducer output will generate an interrupt. The interrupt routine counts transducer pulses, and when it gets to the predetermined N, it reads and clears the timer. For the specific example cited above, the interrupt routine might be:

INTERRUPT_RESPONSE:

```
DJNZ N,OUT
MOV N,#9
CLR EA
CLR TR1
MOV NT_LO,TL1
MOV NT_HI,TH1
MOV TL1,#9
MOV TH1,#0
SETB TR1
SETB EA
CALL LOOKUP_TABLE
OUT: RETI
```

In this routine a pulse counter N is decremented from its preset value, 9, to zero. When the counter gets to zero it is reloaded to 9. Then all interrupts are blocked for a short time while the timer is read and cleared. The timer is stopped during the read and clear operations, so "clearing" it actually means presetting it to 9, to make up for the 9 machine cycles that are missed while the timer is stopped.

The subroutine LOOKUP_TABLE is used to scale the measurement back to the desired 8-bit resolution. It can also include built-in corrections for errors or nonlinearities in the transducer's transfer function.

The subroutine uses the MOVC A, @ A + DPTR instruction to access the table, which contains 270 entries commencing at the 16-bit address referred to as TABLE. The subroutine must compute the address of the table entry that corresponds to the measured value of NT. This address is

$$DPTR = TABLE + NT - NTMIN,$$

where NTMIN = 999, in this specific example.

LOOKUP_TABLE:

```
PUSH ACC
PUSH PSW
MOV A,#LOW(TABLE-NTMIN)
ADD A,NT_LO
MOV DPL,A
MOV A,#HIGH(TABLE-NTMIN)
```

```
ADDC A,NT_HI
MOV DPH,A
CLR A
MOVC A,@A+DTPR
MOV PERIOD,A
POP PSW
POP ACC
RET
```

At this point the value of the period of the transducer signal, measured to 8 bit resolution, is contained in PERIOD.

PULSE WIDTH MEASUREMENTS

The 80C51BH timers have an operating mode which is particularly suited to pulse width measurements, and will be useful in these applications if the transducer signal has a fixed duty cycle.

In this mode the timer is turned on by the on-chip circuitry in response to an input high at the external interrupt pin, and off by an input low, and it can do this while the 80C51BH is in Idle. (The "GATE" mode of timer operation is described in the Intel Microcontroller Handbook.) The external interrupt itself can be enabled, so the same 1-to-0 transition from the transducer that turns off the timer also generates an interrupt. The interrupt routine then reads and resets the timer.

The advantage of this method is that the transducer signal has direct access to the timer gate, with the result that variations in interrupt response time have no effect on the measurement.

Resonant transducers that are designed to fully exploit the GATE mode have an internal divide-by-N circuit that fixes the duty cycle at 50% and lowers the output frequency to the range of 250 to 500 Hz (to control RFI). The transfer function between transducer period and measurand is approximately linear, with known and repeatable error functions.

HMOS/CHMOS Interchangeability

The CHMOS version of the 8051 is architecturally identical with the HMOS version, but there are nevertheless some important differences between them which the designer should be aware of. In addition, some applications require interchangeability between HMOS and CHMOS parts. The differences that need to be considered are as follows:

External Clock Drive: To drive the HMOS 8051 with an external clock signal, one normally grounds the XTAL1 pin and drives the XTAL2 pin. To drive the CHMOS 8051 with an external clock signal, one must drive the XTAL1 pin and leave the XTAL2 pin unconnected. The reason for the difference is that in the

HMOS 8051, it is the XTAL2 pin that drives the internal clocking circuits, whereas in the CHMOS version it is the XTAL1 pin that drives the internal clocking circuits.

There are several ways to design an external clock drive to work with both types. For low clock frequencies (below 6 MHz), the HMOS 8051 can be driven in the same way as the CHMOS version, namely, through XTAL1 with XTAL2 unconnected. Another way is to drive both XTAL1 and XTAL2; that is, drive XTAL1 and use an external inverter to derive from XTAL1 a signal with which to drive XTAL2.

In either case, a 74HC or 74HCT circuit makes an excellent driver for XTAL1 and/or XTAL2, because neither the HMOS nor the CHMOS XTAL pins have TTL-like input logic levels.

Unused Pins: Unused pins of Ports 1, 2 and 3 can be ignored in both HMOS and CHMOS designs. The internal pullups will put them into a defined state. Unused Port 0 pins in 8051 applications can be ignored, even if they're floating. But in 80C51BH applications, these pins should not be left afloat. They can be externally pulled up or down, or they can be internally pulled down by writing 0s to them.

8031/80C31BH designs may or may not need pullups on Port 0. Pullups aren't needed for program fetches, because in bus operations the pins are actively pulled high or low by either the 8031 or the external program memory. But they are needed for the CHMOS part if the Idle or Power Down mode is invoked, because in these modes Port 0 floats.

Logic Levels: If V_{CC} is between 4.5V and 5.5V, an input signal that meets the HMOS 8051's input logic levels will also meet the CHMOS 80C51BH's input logic levels (except for XTAL1/XTAL2 and RST). For the same V_{CC} condition, the CHMOS device will reach or surpass the output logic levels of the HMOS device. The HMOS device will not necessarily reach the output logic levels of the CHMOS device. This is an important consideration if HMOS/CHMOS interchangeability must be maintained in an otherwise CMOS system.

HMOS 8051 outputs that have internal pullups (Ports 1, 2, and 3) "typically" reach 4V or more if I_{OH} is zero, but not fast enough to meet timing specs. Adding an external pullup resistor will ensure the logic level, but still not the timing, as shown in Figure 23. If timing is an issue, the best way to interface HMOS to CMOS is through a 74HCT circuit.

Idle and Power Down: The Idle and Power Down modes exist only on the CHMOS devices, but if one

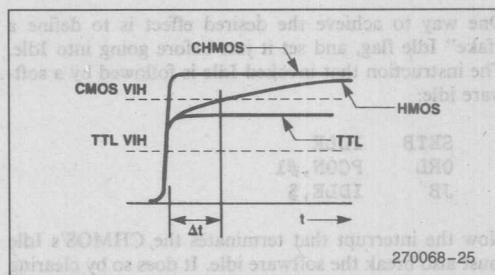


Figure 23. 0-to-1 Transition Shows Unspec'd Delay (Δt) in HMOS to 74HC Logic

wishes to preserve the capability of interchanging HMOS and CHMOS 8051s the software has to be designed so that the HMOS parts will respond in an acceptable manner when a CHMOS reduced power mode is invoked.

For example, an instruction that invokes Power Down can be followed by a "JMP \$":

```
CLR    EA
ORL    PCON, #2
JMP    $
```

The CHMOS and HMOS parts will respond to this sequence of code differently. The CHMOS part, going into a normal CHMOS Power Down Mode, will stop fetching instructions until it gets a hardware reset. The HMOS part will go through the motions of executing the ORL instruction, and then fetch the JMP instruction. It will continue fetching and executing JMP \$ until hardware reset.

Maintaining HMOS/CHMOS 8051 interchangeability in response to Idle requires more planning. The HMOS part will not respond to the instruction that puts the CHMOS part into Idle, so that instruction needs to be followed by a software idle. This would be an idling loop which would be terminated by the same conditions that would terminate the CHMOS's hardware Idle. Then when the CHMOS device goes into Idle, the HMOS version executes the idling loop, until either a hardware reset or an enabled interrupt is received. Now if Idle is terminated by an interrupt, execution for the CHMOS device will proceed after RETI from the instruction following the one that invoked Idle. The instruction following the one that invoked Idle is the idling loop that was inserted for the HMOS device. At this point, both the HMOS and CHMOS devices must be able to fall through the loop to continue execution.

One way to achieve the desired effect is to define a "fake" Idle flag, and set it just before going into Idle. The instruction that invoked Idle is followed by a software idle:

```
SETB   IDLE
ORL     PCON, #1
JB      IDLE, $
```

Now the interrupt that terminates the CHMOS's Idle must also break the software idle. It does so by clearing the "Idle" bit:

```
CLR     IDLE
RETI
```

Note too that the PCON register in the HMOS 8051 contains only one bit, SMOD, whereas the PCON register in CHMOS contains SMOD plus four other bits. Two of those other bits are general purpose flags. Maintaining HMOS/CHMOS interchangeability requires that these flags not be used.

REFERENCES

1. Pawlowski, Moroyan, Alnether, "Inside CMOS Technology," *BYTE magazine*, Sept., 1983. Available as Article Reprint AR-302.
2. Kokkonen, Pashley, "Modular Approach to C-MOS Technology Tailors Process to Application," *Electronics*, May, 1984. Available as Article Reprint AR-332.
3. Williamson, T., *Designing Microcontroller Systems for Electrically Noisy Environments*, Intel Application Note AP-125, Feb. 1982.
4. Williamson, T., "PC Layout Techniques for Minimizing Noise," *Mini-Micro Southeast*, Session 9, Jan., 1984.
5. Alnether, J., *High Speed Memory System Design Using 2147H*, Intel Application Note AP-74, March 1980.
6. Ott, H., "Digital Circuit Grounding and Interconnection," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility*, pp. 292-297, Aug. 1981.
7. *Digital Sensors by Technar*, Technar Inc., 205 North 2nd Ave., Arcadia, CA 91006.

The serial port on the 8051 has been enhanced on the 83C51FA with the addition of two new features: Automatic Address Recognition and Framing Error Detection. **Automatic Address Recognition** facilitates multiprocessor communications by reducing CPU overhead. **Framing Error Detection** increases communication reliability by checking each reception for a valid stop bit.

This Application Note explains how to use these new features with samples of code for typical applications. A section is also included which reviews how to set up the serial port for multiprocessor applications.

MULTIPROCESSOR COMMUNICATIONS

In applications where multiple controllers jointly perform a task, the master controller must be able to communicate selectively with individual slaves. To do this, the master first identifies the target slave (or slaves) with an address byte and then transmits a block of data. The target slaves must be able to identify their own address before receiving any data bytes.

The serial port on the 8051 provides a 9-bit mode to facilitate multiprocessor communication. The 9th bit allows the controller to distinguish between address and data bytes. In this mode, a total of 11 bits are received or transmitted: a start bit (0), 8 data bits (LSB first), a programmable 9th bit, and a stop bit (1). See Figure below.

The 9th bit is set to 1 to identify address bytes and set to 0 for data bytes. A typical data stream is seen below:

ADDRESS BYTE	/	DATA BYTE	/	DATA BYTE	/	...
D8 = 1		D8 = 0		D8 = 0		

Initially the slave is set up to only receive address bytes. Once it receives its own address, the slave reconfigures itself to receive data. On the 8051 serial port, an address byte interrupts **all** slaves for an address comparison. On the 83C51FA, however, Automatic Address Recognition allows the addressed slave to be the **only** one interrupted; that is, the address comparison occurs in hardware, not software. With this feature, the master controller can establish communication with one or more slaves without all the slaves having to respond to the transmission.

AUTOMATIC ADDRESS RECOGNITION

Automatic Address Recognition reduces the CPU time required to service the serial port. Since the CPU is only interrupted when it receives its own address, the software overhead to compare addresses is eliminated. This would also effectively reduce the sophistication of the serial protocol when numerous controllers are sharing the same serial link.

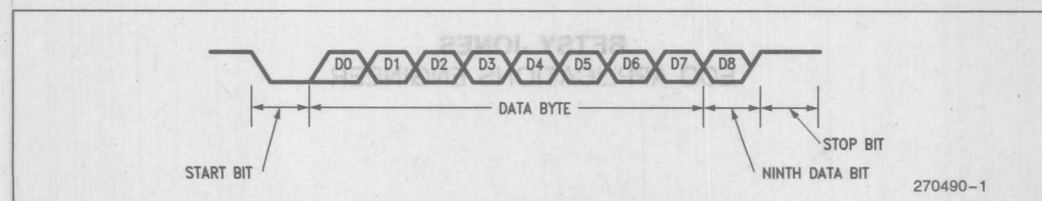
This same feature can also be used in conjunction with the Idle Mode to reduce the system's overall power consumption. For instance, a master may need to communicate with only one slave at a time. With all slaves in Idle Mode, only that one slave would be interrupted to respond to the master's transmission. Without Automatic Addressing, each slave would have to "wake up" to check for its address. Limiting the interruptions reduces the amount of current drawn by the system and thus reduces the power consumption.

In multiprocessor applications the serial port is configured in either of the 9-bit modes (Mode 2 or 3). Mode 2 has a fixed baud rate whereas Mode 3 is variable. For more information on the different serial port modes refer to the "Serial Port Set Up" section.

Automatic Address Recognition is enabled by setting the SM2 bit in SCON. Each slave has its SM2 bit set waiting for an address byte (9th bit = 1). The Receive Interrupt (RI) flag will get set when the received byte corresponds to either a Given or Broadcast Address. The slave then clears its SM2 bit to enable reception of data bytes (9th bit = 0) from the master.

The master can selectively communicate with groups of slaves by using the Given Address. Addressing all slaves at once is possible with the Broadcast Address. These addresses are defined for each slave by two new Special Function Registers: SADDR and SADEN.

A slave's individual address is specified in SADDR. SADEN is a mask byte that defines don't-cares to form the Given Address. These don't-cares allow flexibility in the user-defined protocol to address one or more slaves. The following is an example of how to define Given Addresses and selectively address different slaves.



Slave 1			
SADDR	=	1111	0001
SADEN	=	1111	1010
GIVEN	=	1111	0X0X
Slave 2			
SADDR	=	1111	0011
SADEN	=	1111	1001
GIVEN	=	1111	0XX1

The SADEN bytes have been selected such that bit 1 (LSB) is a don't-care for Slave 1's Given Address, but bit 1 = 1 for Slave 2. Thus, to selectively communicate with just Slave 1 an address with bit 1 = 0 would be used (e.g. 1111 0000).

Similarly, bit 2 = 0 for Slave 1, but is a don't-care for Slave 2. Now to communicate with just Slave 2 an address with bit 2 = 1 would be used (e.g. 1111 0111).

Finally, to communicate with both slaves at once the address must have bit 1 = 1 and bit 2 = 0. Notice, however, that bit 3 is a "don't-care" for both slaves. This allows two different addresses to select both slaves (1111 0001 or 1111 0101). If a third slave was added that required its bit 3 = 0, then the latter address could be used to communicate with Slave 1 and 2 but not Slave 3.

The master can also communicate with all slaves at once with the Broadcast Address. It is formed from the logical OR of SADDR and SADEN with zeros defined as don't-cares. For example, the Broadcast address for Slave 1 would be formed as follows:



The don't-cares also allow flexibility in defining the Broadcast Address, but in most applications a Broadcast Address will be 0FFH.

SADDR and SADEN are located at address A9H and B9H, respectively. On Reset, SADDR and SADEN are initialized to 00H which defines the Given and Broadcast Addresses as XXXX XXXX (all don't-cares). This assures the 83C51FA serial port to be backwards compatible with the other MCS®-51 products which do not implement Automatic Addressing.

FRAMING ERROR DETECTION

Framing Error Detection is another new feature on 83C51FA serial port which allows the receiving controller to check for valid stop bits in Modes 1, 2, or 3. A missing stop bit can be caused, for example, by noise on the serial lines or transmission by two CPUs simultaneously.

If a stop bit is missing a Framing Error bit FE will be set. This bit can then be checked in software after each reception to detect communication errors. Once set, the FE bit must be cleared in software. A valid stop bit will not clear FE.

The FE bit is located in SCON and shares the same bit address as SM0. To determine which is accessed, a new control bit called SMOD0 has been added in the PCON register (see figures below). If SMOD0 = 0, then accesses to SCON.7 are to SM0. If SMOD0 = 1, then accesses to SCON.7 are to FE.

PCON: Power Control Register (Not Bit Addressable)

SMOD1	SMOD0	—	POF	GF1	GF0	PD	IDL
-------	-------	---	-----	-----	-----	----	-----

Address = 87H

SCON: Serial Port Control Register (Bit Addressable)

SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
--------	-----	-----	-----	-----	-----	----	----

Address = 98H

SERIAL PORT SOFTWARE

The following sections of code show examples of how to invoke Automatic Addressing and Framing Error Detection. Routines for both the slave and master are given. Code is also included to initialize both serial ports; however, for more information on setting up the serial port refer to the next section.

For this example, the master and slave are transmitting/receiving at 9600 baud with a 12 MHz crystal frequency. To obtain this baud rate, the serial port is configured in Mode 3 and Timer 2 is used as the baud-rate generator.

Listing 1 shows the initialization for the slave. Notice that Automatic Addressing and Framing Error Detection are enabled. The Given and Broadcast addresses for this slave are taken from Slave 1 in the previous example. A temporary byte has also been defined to store the incoming data byte.

The slave will remain in Idle Mode until it is interrupted by its own address. At that point, it clears the SM2

Listing 1. Initialization Routine for the Slave

```

ORG 00H
LJMP INIT

ORG 0023H
LJMP SERIAL_PORT_INTERRUPT

TEMP DATA 30H ; Temporary storage byte

INIT: MOV SCON, #0FOH ; Mode 3, enable Auto Addressing
      ; and reception
      ORL PCON, #40H ; FE bit accessed (SMOD0 = 1)
      MOV RCAP2H, #OFFH ; Reload values for 9600 Baud
      MOV RCAP2L, #0D9H
      MOV T2CON, #34H ; Timer 2 set up, TR2 = 1 turns
                        ; timer on

INTERRUPTS: SETB EA ; Enable global interrupt
            SETB ES ; Enable serial port interrupt

ADDRESSES: MOV SADDR, # 11110001 ; Define Given & Broadcast
            MOV SADEN, # 11111010 ; Addresses
            ; GIVEN = 11110X0X
            ; BROADCAST = 11111X11

IDLE_MODE: ORL PCON, #01H ; Invoke Idle Mode

```

Listing 2. Receive Routine for the Slave

```

SERIAL_PORT_INTERRUPT:
    PUSH PSW
    CLR RI ; RI set when address is
           ; recognized & must be cleared
           ; in software
    CLR SM2 ; Reconfigure slave to receive
           ; data bytes

RECEIVE_DATA:
    JNB RI, $ ; Wait for RI to be set
    MOV C, SCON.7 ; Check for framing error
    JC FRAMING_ERROR
    MOV TEMP, SBUF ; Receive data byte & store
                  ; in temporary location
    CLR RI ; Clear flag for next
           ; reception
    SETB SM2 ; Re-enable Automatic
           ; Addressing

    POP PSW
    RETI

FRAMING_ERROR:
    CLR SCON.7 ; Clear FE bit
    CLR C
    ; Error routine left up to
    ; the user

    POP PSW
    RETI

```

Listing 3. Initialization and Transmit Routines for the Master

```

GIVEN_1      equ      11110001B
MESSAGE_1    data     30H

INIT:  MOV SCON, #0D0H      ; Mode 3, REN = 1
      MOV RCAP2H, #0FFH    ; 9600 Baud
      MOV RCAP2L, #0D9H    ; Timer 2 set up, TR2 = 1
      MOV T2CON, #34H

TRANSMIT_ADDRESS:
      CLR TI                ; Mark 1st byte as an address
      SETB TB8              ; byte (9th bit = 1)
      MOV SBUF, #GIVEN_1    ; Send address
      JNB TI, $             ; Wait for transmission
                              ; complete
      CLR TI                ; Clear flag for next
                              ; transmission

TRANSMIT_DATA:
      CLR TB8              ; Mark 2nd byte as a data
                              ; byte (9th bit = 0)
      MOV SBUF, MESSAGE_1   ; Send data byte
      JNB TI, $
      CLR TI

```

bit to enable reception of data bytes. Depending on the user's protocol, more than one data byte may actually be received. This example, however, assumes only one byte of data follows each address byte.

Listing 2 shows the receive routine. Notice that when the data byte is received, the software checks for a framing error. The error routine could, for example, send an error message to the master and ask the master to re-transmit the last message. Before exiting the routine the SM2 is set to 1 to reenble Automatic Addressing. Once the slave has responded to the master's command, it could also put itself back into Idle Mode to wait for the next message.

The initialization routine for the master in Listing 3 is very similar to the slave. In this example, however, the master does not need Automatic Addressing; it is simply transmitting address and data bytes. GIVEN_1 is a byte to address the slave in the above example. MESSAGE_1 is a register that contains the data byte sent to this slave. Its value is arbitrary for the sample code.

SERIAL PORT SET UP

This section describes how to initialize the 83C51FA serial port for multiprocessor applications. Two different modes are available which provide 9-bit operation:

Mode 2 which has a fixed baud rate and Mode 3 which has a variable baud rate. Baud rates can be generated by either Timer 1 or Timer 2 (available on the 83C51FA but not the 8051). Deciding which mode and timer to use is determined by the desired baud rate and clock frequency of the particular application.

Another consideration is the tolerance needed between serial ports. Since the serial port re-synchs its receiver at every start bit, only 8 or 9 bit-times are available to accumulate timing errors. As a result, the receiver and transmitter only have to be within about 5% of each other's baud rate. Allowing equal error to both transmitter and receiver, only about 2% accuracy is actually needed.

Following is a discussion of both Modes 2 and 3 and examples of how to program each. The mode selection bits (SM0 and SM1) are located in SCON. The REN bit must also be set to enable reception.

SCON: Serial Port Control Register (Bit Addressable)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
-----	-----	-----	-----	-----	-----	----	----

Address = 98H

Mode	SM0	SM1	Baud Rate
2	1	0	Fosc/64 or Fosc/32
3	1	1	Variable

Example 1. Serial Port Mode 2

```
; Frequency      = 12 MHz
; Desired Baud Rate = 375 kBaud
;               = 1/32 (Osc Freq)

MOV SCON, #0B0H ; Serial port Mode 2
                ; Automatic Addressing (SM2 = 1),
                ; reception enabled (REN = 1)
ORL PCON, #080H ; SMOD1 = 1 to double baud rate
```

Mode 2

Mode 2 uses a fixed baud rate of 1/32 or 1/64 of the oscillator frequency depending on the value of the SMOD1 bit in PCON. This mode basically offers a choice of two high-speed baud rates. With a 12 MHz clock frequency, baud rates of 187.5 kbaud or 375 kbaud can be obtained.

None of the timer/counters need to be set up for Mode 2. Only the SFRs SCON and PCON need to be defined.

PCON: Power Control Register (Not Bit Addressable)

SMOD1	SMOD0	—	POF	GF1	GF0	PD	IDL
-------	-------	---	-----	-----	-----	----	-----

Address = 87H

The baud rate in this mode is calculated by:

$$\text{Mode 2 Baud Rate} = \frac{2^{\text{SMOD1}} \times \text{Osc Freq}}{64}$$

SMOD1 = 0, Baud Rate = 1/64 Osc Freq

SMOD1 = 1, Baud Rate = 1/32 Osc Freq

Mode 3

Mode 3 of the serial port has a variable baud rate generated by either Timer 1 or Timer 2. The baud rate is generated by the rollover rate of the selected timer. The timer is operated in an auto-reload mode so it will roll over to the reload value selected in software.

Baud rates based off Timer 2 have less granularity so that almost any baud rate can be obtained at a given clock frequency. However, Timer 1 is sufficient if the desired baud rate can be obtained at the specified clock frequency. Remember baud rates only need about 2% accuracy.

Timer 1 Set Up

To generate baud rates Timer 1 is usually configured in 8-bit auto-reload mode (Mode 2). The mode select bits

are M1 and M0 located in TMOD. To turn on Timer 1 the TR1 bit in TCON must be set. Also, the Timer 1 interrupt should be disabled in this application so that when the timer overflows it does not generate an interrupt.

TMOD: Timer/Counter Mode Control Register
(Not bit addressable)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				Timer 0			

Address = 89H

TCON: Timer/Counter Control Register
(Bit addressable)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
-----	-----	-----	-----	-----	-----	-----	-----

Address = 88H

The formula for calculating the baud rate is given below. TH1 is the reload value for Timer 1 when it overflows.

$$\text{Baud Rate} = \frac{K \times \text{Osc Freq}}{32 \times 12 \times [256 - (\text{TH1})]}$$

K = 1 if SMOD1 = 0.

K = 2 if SMOD1 = 1. (SMOD1 is at PCON.7)

If the baud rate is known, the reload value TH1 can be calculated by:

$$\text{TH1} = 256 - \frac{K \times \text{Osc Freq}}{384 \times \text{Baud Rate}}$$

TH1 must be an integer value. Rounding off TH1 to the nearest integer may not produce the desired baud rate with the 2% accuracy required. In this case, another crystal frequency may have to be chosen.

Refer to Table 1 for timer reload values for commonly used baud rates.

Table 1. Commonly Used Baud Rates Generated by Timer 1

Baud Rate	Osc Freq	SMOD1	Timer 1	
			TMOD	Reload Value
62.5K	12 MHz	1	20	FFH
19.2K	11.06 MHz	1	20	FDH
9.6K	11.06 MHz	0	20	FDH
4.8K	11.06 MHz	0	20	FAH
2.4K	11.06 MHz	0	20	F4H
1.2K	11.06 MHz	0	20	E8H
300	6 MHz	0	20	CCH
110	6 MHz	0	20	72H

Example 2. Serial Port Mode 3, with Timer 1 as Baud-Rate Generator

```

; Frequency = 11.0 MHz
; Desired Baud Rate = 19.2 kBaud
;
; TH1 = 256 - ((2) x (11.0 x 106)) / ((32) x (12) x (19200))
;
; = 253 = FDH

```

```

MOV SCON, #0FOH ; Serial port Mode 3, SM2 = 1,
; REN = 1
ORL PCON, #80H ; SMOD1 = 1
MOV TMOD, #20H ; Timer 1 Mode 2
MOV TH1, #0FDH ; Reload value for desired baud
; rate
SETB TR1 ; Turn on Timer 1

```

It can be seen that the exact frequency to generate the standard baud rates (19.2K, 9600, 4800, etc.) is 11.06 MHz. However, it is not necessary to use this exact frequency. With a 2% tolerance any crystal value from 10.8 MHz to 11.3 MHz is sufficient.

Timer 2 Set Up

Timer 2 has a special baud-rate generator mode which transmits and receives at the same baud rate. This mode is invoked by setting both the RCLK and TCLK bits in T2CON. To turn Timer 2 on the TR2 bit should also be set.

Unlike Timer 1, this mode does not require that the timer overflow interrupt be disabled. That is, when Timer 2 is in the baud-rate generator mode, its interrupt is disconnected from the Timer 2 overflow. This

interrupt then becomes available as a third external interrupt. (For more information on external interrupts, refer to the chapter "Hardware Description of the 8051" in the Embedded Controller Handbook.)

T2CON: Timer/Counter 2 Control Register
(Bit Addressable)

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2
-----	------	------	------	-------	-----	------	--------

Address = C8H

This formula for calculating the baud rate is given below. (RCAP2H, RCAP2L) is the 16-bit reload value when Timer 2 overflows.

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{32 \times [65536 - (\text{RCAP2H}, \text{RCAP2L})]}$$

where (RCAP2H, RCAP2L) is a 16-bit unsigned integer.

To obtain the reload value for RCAP2H and RCAP2L the above equation can be rewritten as:

$$(RCAP2H, RCAP2L) = 65536 - \frac{\text{Osc Freq}}{32 \times \text{Baud Rate}}$$

Refer to Table 2 for reload values for commonly used baud rates.

Notice that when using Timer 2, most standard baud rates can be obtained at 12 MHz.

Table 2. Commonly Used Baud Rates Generated by Timer 2

Baud Rate	Osc Freq	Timer 2	
		RCAP2H	RCAP2L
375K	12 MHz	FF	FF
9.6K	12 MHz	FF	D9
4.8K	12 MHz	FF	B2
2.4K	12 MHz	FF	64
1.2K	12 MHz	FE	C8
300	12 MHz	FB	1E
110	12 MHz	F2	AF
300	6 MHz	FD	8F
110	6 MHz	F9	57

Example 3. Serial Port Timer with Timer 2 as Baud-Rate Generator

```

; Frequency          = 12 MHz
; Desired Baud Rate  = 9600 Baud
;
; (RCAP2H, RCAP2L) = 65536 - (12 x 106) / (32 x (9600))
;
; = 65497 = FFD9H
;
MOV SCON, #0F0H      ; Serial port Mode 3, SM2 = 1,
                     ; REN = 1
MOV RCAP2H, #0FFH    ; Reload values for desired
MOV RCAP2L, #0D9H    ; baud rate
MOV T2CON, #34H      ; Timer 2 as baud rate
                     ; generator, turn on Timer 2
    
```

interrupt then becomes available as a third external interrupt. For more information on external interrupts, refer to the chapter "Hardware Description of the 8051" in the Embedded Controller Handbook.

T2CON: Timer/Counter 2 Control Register
(Bit Addressable)

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	CP/RL2
-----	------	------	------	-------	-----	--------

Address = C8H

This formula for calculating the baud rate is given below. (RCAP2H, RCAP2L) is the 16-bit reload value when Timer 2 overflows.

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{32 \times (65536 - (RCAP2H, RCAP2L))}$$

where (RCAP2H, RCAP2L) is a 16-bit unsigned integer.

It can be seen that the exact frequency to generate the standard baud rates (19.2K, 9600, 4800, etc.) is 11.06 MHz. However, it is not necessary to use this exact frequency. With a 1% tolerance any crystal value from 10.8 MHz to 11.3 MHz is sufficient.

Timer 2 Set Up

Timer 2 has a special baud-rate generator mode which transmits and receives at the same baud rate. This mode is invoked by setting both the RCLK and TCLK bits in T2CON. To turn Timer 2 on the TR2 bit should also be set.

Unlike Timer 1, this mode does not require that the timer overflow interrupt be disabled. That is, when Timer 2 is in the baud-rate generator mode its interrupt is disconnected from the Timer 2 overflow. This



APPLICATION BRIEF

AB-41

PAGE

S-223

S-223

S-223

S-223

S-223

S-230

S-231

Introduction

Variables

Initialization

Receive Routine

Transmit Routine

Conclusion

APPENDIX

SOFTWARE SERIAL PORT
IMPLEMENTED WITH THE
PCA

April 1989

2

Software Serial Port Implemented with the PCA

BETSY JONES
ECO APPLICATIONS ENGINEER

Order Number: 270531-002

AB-41 APPLICATION BRIEF

SOFTWARE SERIAL PORT IMPLEMENTED WITH THE PCA

CONTENTS

PAGE

Introduction	2-223
Variables	2-223
Initialization	2-225
Receive Routine	2-225
Transmit Routine	2-229
Conclusion	2-230
APPENDIX	2-231

April 1989

2

Software Serial Port Implemented
with the PCA

BETSY JONES
ECO APPLICATIONS ENGINEER

Order Number: 570531-005

For microcontroller applications which require more than one serial port, the 83C51FA Programmable Counter Array (PCA) can implement additional half-duplex serial ports. If the on-chip UART is being used as an inter-processor link, the PCA can be used to interface the 83C51FA to additional asynchronous lines.

This application uses several different Compare/Capture modes available on the PCA to receive or transmit bytes of data. It is assumed the reader is familiar the PCA and ASM51. For more information on the PCA refer to the "Hardware Description of the 83C51FA" chapter in the Embedded Controller Handbook (Order No. 210918).

Introduction

The figure below shows the format of a standard 10-bit asynchronous frame: 1 start bit (0), 8 data bits, and 1 stop bit (1). The start bit is used to synchronize the receiver to the transmitter; at the leading edge of the start bit the receiver must set up its timing logic to sample the incoming line in the center of each bit. Following the start bit are eight data bits which are transmitted least significant bit first. The stop bit is set to the opposite state of the start bit to guarantee that the leading edge of the start bit will cause a transition on the line. It also provides a dead time on the line so that the receiver can maintain its synchronization.

Two of the Compare/Capture modes on the PCA are used in receiving and transmitting data bits. When receiving, the Negative-Edge Capture mode allows the PCA to detect the start bit. Then using the Software Timer mode, interrupts are generated to sample the incoming data bits. This same mode is used to clock out bits when transmitting.

This Application Note contains four sections of code:

- (1) List of variables
- (2) Initialization routine

- (3) Receive routine
- (4) Transmit routine.

A complete listing of the routines and the test loop which was used to verify their operation is found in the Appendix. A total of three half-duplex channels were run at 2400 Baud in the test program. The listings shown here are simplified to one channel (Channel 0).

Variables

Listing 1 shows the variables used in both the receive and transmit routines. Flags are defined to signify the status of the reception or transmission of a byte (e.g. RCV_START_BIT, TXM_START_BIT). RCV_BUF and TXM_BUF simulate the on-chip serial port SBUF as two separate buffer registers. The temporary registers, RCV_REG and TXM_REG, are used to save bits as they are received or transmitted. Finally, two counter registers keep track of how many bits have been received or transmitted.

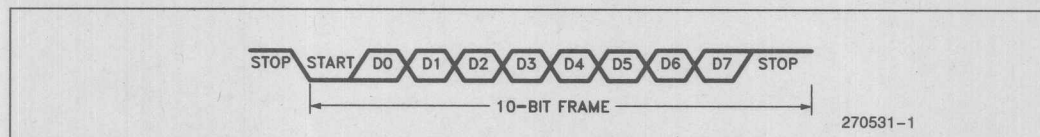
Variables are also needed to define one-half and one-full bit times in units of PCA timer ticks. (One bit time = 1 / baud rate.) With the PCA timer incremented every machine cycle, the equation to calculate one bit time can be written as:

$$\frac{\text{Osc. Freq.}}{(12) \times (\text{baud rate})} = 1 \text{ bit time (in PCA timer ticks)}$$

In this example, the baud rate is 2400 at 16 MHz.

$$\frac{16 \text{ MHz}}{(12) \times (2400)} = 556 \text{ counts} = 22\text{C Hex}$$

The high and low byte of this value is placed in the variables FULL_BIT_HIGH and FULL_BIT_LOW, respectively. 115H is the value loaded into HALF_BIT_HIGH and HALF_BIT_LOW.



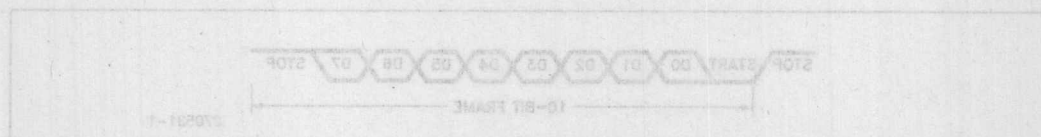
Listing 1. Variables used by the software serial port. Channel 0

; <u>Receive Routine</u>			
RCV_START_BIT_0	BIT	20H.0	; Indicates start bit
			; has been received
RCV_DONE_0	BIT	20H.1	; Indicates data byte
			; has been received
RCV_BUF_0	DATA	30H	; Software Receive
			; "SBUF"
RCV_REG_0	DATA	31H	; Temporary register
			; for receive bits
RCV_COUNT_0	DATA	32H	; Counter for receiving
			; bits
; <u>Transmit Routine:</u>			
TXM_START_BIT_0	BIT	20H.3	; Indicates start bit
			; has been transmitted
TXM_IN_PROGRESS_0	BIT	20H.4	; Indicates transmit is
			; in progress
TXM_BUF_0	DATA	34H	; Software transmit
			; "SBUF"
TXM_REG_0	DATA	35H	; Temporary register
			; for transmitting bits
TXM_COUNT_0	DATA	36H	; Counter for transmit-
			; ting bits
DATA_0	DATA	37H	; Register used for the
			; test program
;			
NEG_EDGE	EQU	11H	; Two modes of operation
S_W_TIMER	EQU	49H	; for compare/capture
			; modules
;			
HALF_BIT_HIGH	EQU	01H	; Half bit time = 115H
HALF_BIT_LOW	EQU	15H	
FULL_BIT_HIGH	EQU	02H	; Full bit time = 22CH
FULL_BIT_LOW	EQU	2CH	; 2400 Baud at 16 MHz

270531-4

The high and low byte of this value is placed in the variables FULL_BIT_HIGH and FULL_BIT_LOW respectively. 115H is the value loaded into HALF_BIT_HIGH and HALF_BIT_LOW.

(1) List of variables
(2) Initialization routine



Initialization

Listing 2 contains the initialization code for the receive and transmit process. Module 0 of the PCA is used as a receiver and is first set up to detect a negative edge from the start bit. Modules 2 and 3 are used for the additional 2 channels (see the Appendix). Module 3 is used as a separate software timer to transmit bits.

Listing 2. Initialization Routine

```

ORG 0000H
LJMP INITIALIZE
ORG 001BH
LJMP RECEIVE_DONE
; Timer 1 overflow -
; simulates "RI" interrupt

ORG 0033H
LJMP RECEIVE
; PCA interrupt

; INITIALIZE: MOV SP, #5FH
; Initialize stack pointer
; (specific to test program)
INIT_PCA: MOV CMOD, #00H
; Increment PCA timer
; @ 1/12 Osc Frequency
MOV CCON, #00H
; Clear all status flags
MOV CCAPM0, #NEG_EDGE
; Module 0 in negative-edge
; trigger mode (P1.3)
MOV CCAPM3, #S_W_TIMER
; Module 3 as software timer
; mode

MOV CL, #00H
MOV CH, #00H
MOV IE, #0D8H
; Init all needed interrupts
; EA, EC, ES, ET1
SETB CR
; Turn on PCA Counter

```

270531-5

All flags and registers from Listing 1 should be cleared in the initialization process.

Receive Routine

Two operating modes of the PCA are needed to receive bits. The module must first be able to detect the leading edge of a start bit so it is initially set up to capture a 1-to-0 transition (i.e. Negative-Edge Capture mode). The module is then reconfigured as a software timer to cause an interrupt at the center of each bit to deserialize the incoming data. The flowchart for the receive routine is given in Figure 1.

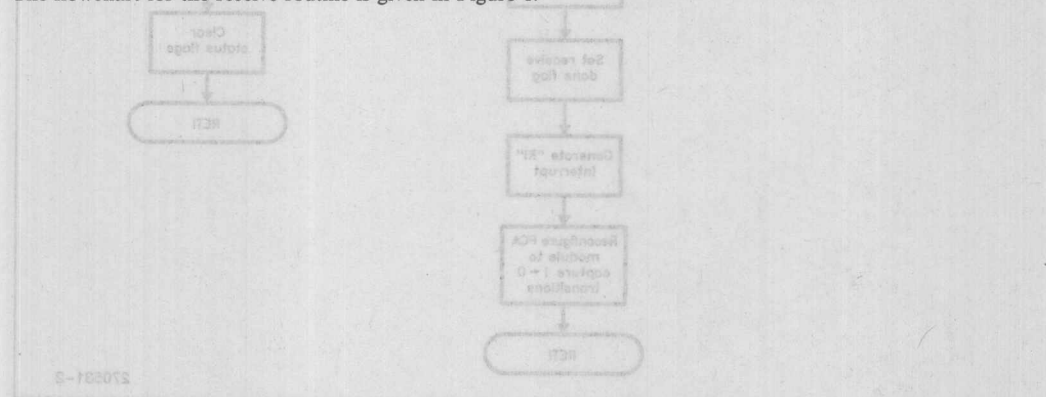


Figure 1. Flowchart for the Receive Routine

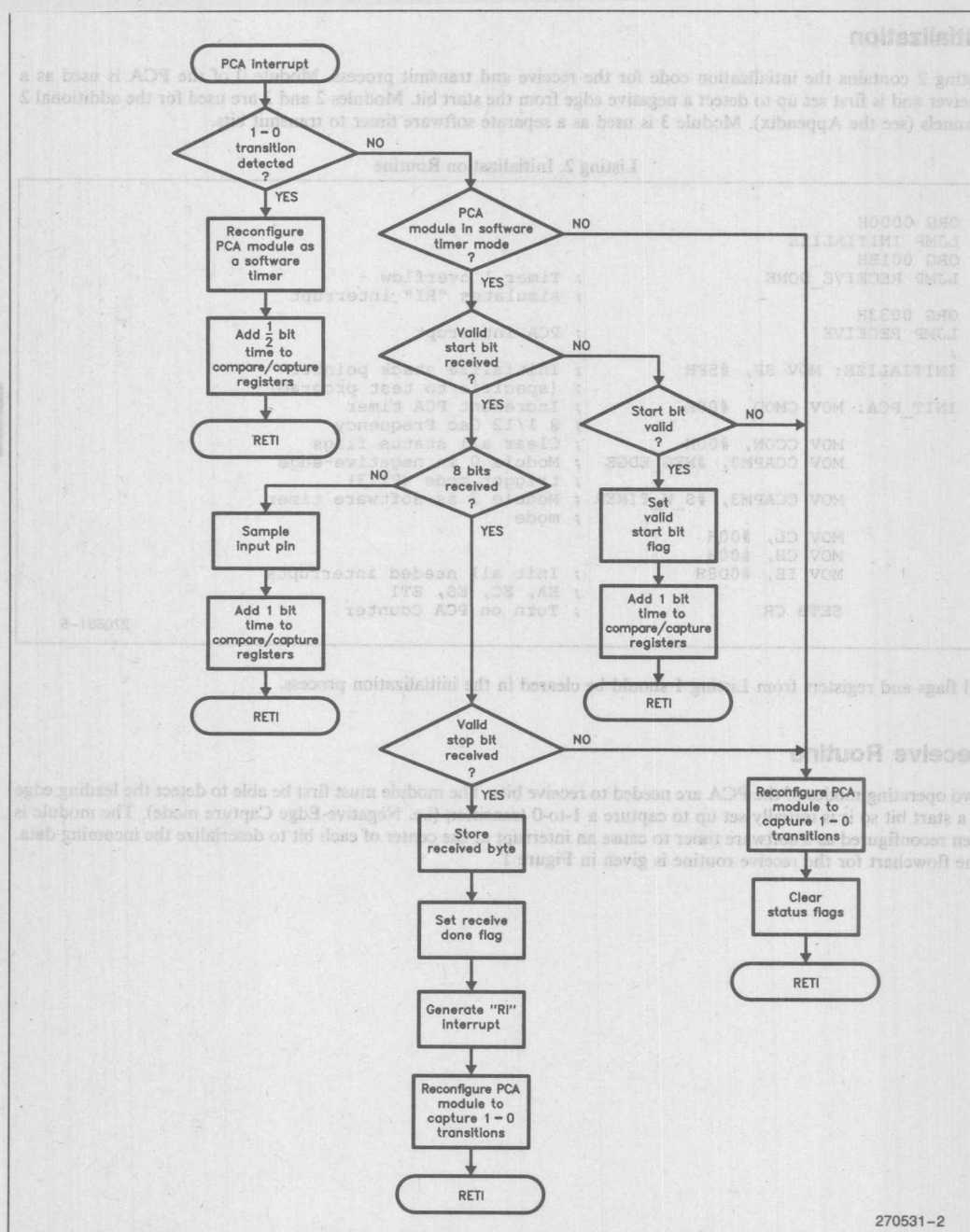


Figure 1. Flowchart for the Receive Routine

Listing 3.1 shows the code needed to detect a start bit. Notice that the first software timer interrupt will occur one-half bit time after the leading edge of the start bit to check its validity. If it is valid, the RCV_START_BIT is set. The rest of the samples will occur a full bit time later. The RCV_COUNT register is loaded with a value of 9 which indicates the number of bits to be sampled: 8 data bits and 1 stop bit.

Listing 3.1. Receive Interrupt Routine

```

RECEIVE:  PUSH ACC
          PUSH PSW
;
MODULE_0: CLR CCF0           ; Assume reception on
          ; Module 0
          MOV A, CCAPM0       ; Check mode of module. If
          ANL A, #01111111B    ; set up to receive negative
          CJNE A, #NEG_EDGE, RCV_START_0 ; edges, then module
          ; is waiting for a start bit
;
          CLR C               ; Update compare/capture
          MOV A, #HALF_BIT_LOW ; registers for half bit time
          ADD A, CCAP0L        ; to sample start bit
          MOV CCAP0L, A        ; Half bit time = 115H
          MOV A, #HALF_BIT_HIGH
          ADDC A, CCAP0H
          MOV CCAP0H, A
          MOV CCAPM0, #S_W_TIMER ; Reconfigure module 0 as
          POP PSW              ; a software timer to sample
          POP ACC              ; bits
          RETI
;
RCV_START_0: CJNE A, #S_W_TIMER, ERROR_0 ; Check module is
          ; configured as a software
          ; timer, otherwise error.
          JB RCV_START_BIT_0, RCV_BYTE_0 ; Check if start bit
          ; is Received yet.
          JB P1.3, ERROR_0           ; Check that start bit = 0,
          ; otherwise error.
          SETB RCV_START_BIT_0       ; Signify valid start bit
          ; was received
          MOV RCV_COUNT_0, #09H      ; Start counting bits sampled
;
          CLR C                       ; Update compare/capture
          MOV A, #FULL_BIT_LOW        ; registers to sample
          ADD A, CCAP0L                ; incoming bits
          MOV CCAP0L, A                ; Full bit time = 22CH
          MOV A, #FULL_BIT_HIGH
          ADDC A, CCAP0H
          MOV CCAP0H, A
          POP PSW
          POP ACC
          RETI

```

2

270531-6

The next 8 timer interrupts will receive the incoming data bits; the RCV_COUNT register keeps track of how many bits have been sampled. As each bit is sampled, it is shifted through the Carry Flag and saved in RCV_REG. The ninth sample checks the validity of the stop bit. If it is valid, the data byte is moved into RCV_BUF.

The main routine must have a way to know that a byte has been received. With the on-chip UART, the RI (Receive Interrupt) bit is set whenever a byte has been received. For the software serial port, any unimplemented interrupt vector can be used to generate an interrupt when a byte has been received. This routine uses the Timer 1 Overflow interrupt (its selection is arbitrary). A routine to test this interrupt is included in the listing in the Appendix.

Listing 3.2. Receive Interrupt Routine (Continued)

```
RCV_BYTE_0: DJNZ RCV_COUNT_0, RCV_DATA_0 ; On 9th sample,
; check for valid stop bit
RCV_STOP_0: JNB P1.3, ERROR_0
MOV RCV_BUF_0, RCV_REG_0 ; Save received byte in
; receive "SBUF"
SETB RCV_DONE_0 ; Flag which module received
; a byte
SETB TF1 ; Generate an interrupt so
; main program knows a byte
; has been received
; (Note: selection of TF1 is
; arbitrary)
MOV CCAPM0, #NEG_EDGE ; Reconfigure module 0 for
; Reception of a start bit
POP PSW
POP ACC
RETI

;
RCV_DATA_0: MOV C, P1.3 ; Sampling data bits
MOV A, RCV_REG_0 ; Shifts bits thru CY into
RRC A ; ACC
MOV RCV_REG_0, A ; Save each reception in
; temporary register
CLR C ; Update c/c register for
; next sample time
MOV A, #FULL_BIT_LOW
ADD A, CCAP0L
MOV CCAP0L, A
MOV A, #FULL_BIT_HIGH
ADDC A, CCAP0H
MOV CCAP0H, A
POP PSW
POP ACC
RETI
```

270531-7

In addition, an error routine (Listing 3.3) is included for invalid start or stop bits to offer some protection against noise. If an error occurs, the module is re-initialized to look for another start bit.

Listing 3.3 Error Routine for Receive Routine

```
ERROR_0: MOV CCAPM0, #NEG_EDGE ; Reset module to look for
; start bit
CLR RCV_START_BIT_0 ; Clear flags which might
; have been set
POP PSW
POP ACC
RETI
```

270531-8

Transmit Routine

Another PCA module is configured as a software timer to interrupt the CPU every bit time. With each timer interrupt one or more bits can be transmitted through port pins. In the test program three channels were operated simultaneously, but in the listings below, one channel is shown for simplicity. The selection of port pins is user programmable. The flowchart for the transmit routine is given in Figure 2.

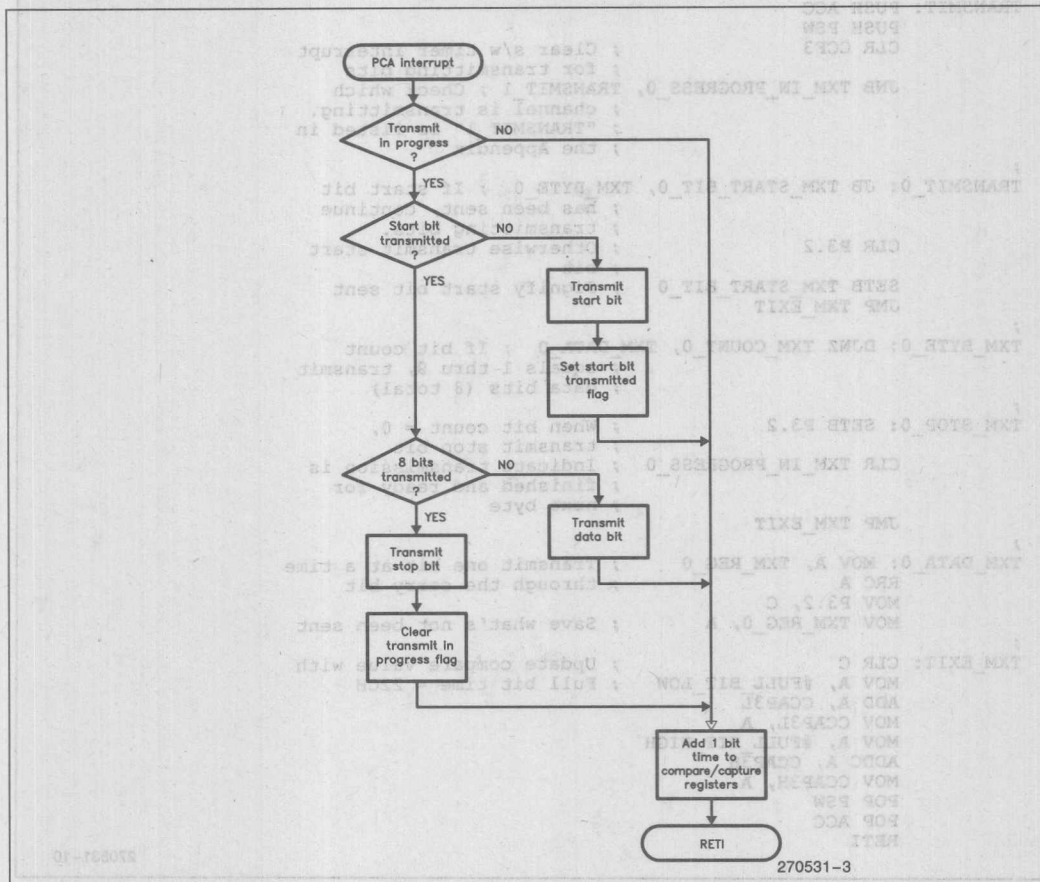


Figure 2. Flowchart for the Transmit Routine

When a byte is ready to be transmitted, the main program moves the data byte into the TXM_BUF register and sets the corresponding TXM_IN_PROGRESS bit. This bit informs the interrupt routine which channel is transmitting. The data byte is then moved in the storage register TXM_REG, and the TXM_COUNT is loaded. This main routine is shown in Listing 4.1.

Listing 4.1 Transmit Set Up Routine. Channel 0.

```

TXM_ON_0: CLR TXM_START_BIT_0 ; Clear status flag from
                                ; previous transmission
MOV TXM_BUF_0, DATA_0 ; Load "SBUF" with data byte
MOV TXM_REG_0, TXM_BUF_0
MOV TXM_COUNT_0, #09 ; 8 data bits + 1 stop bit
SETB TXM_IN_PROGRESS_0
  
```

270531-9

Listing 4.2 shows the transmit interrupt routine. The first time through, the start bit is transmitted. As each successive interrupt outputs a bit, the contents of TXM_REG is shifted right one place into the Carry flag, and the TXM_COUNT is decremented. When TXM_COUNT equals zero, the stop bit is transmitted.

Listing 4.2. Transmit Interrupt Routine

```

TRANSMIT: PUSH ACC
          PUSH PSW
          CLR CCF3                                ; Clear s/w timer interrupt
                                                  ; for transmitting bits
          JNB TXM_IN_PROGRESS_0, TRANSMIT_1      ; Check which
                                                  ; channel is transmitting.
                                                  ; "TRANSMIT 1" is listed in
                                                  ; the Appendix

;
TRANSMIT_0: JB TXM_START_BIT_0, TXM_BYTE_0      ; If start bit
                                                  ; has been sent, continue
                                                  ; transmitting bits.
          CLR P3.2                                ; Otherwise transmit start
                                                  ; bit
          SETB TXM_START_BIT_0                  ; Signify start bit sent
          JMP TXM_EXIT

;
TXM_BYTE_0: DJNZ TXM_COUNT_0, TXM_DATA_0        ; If bit count
                                                  ; equals 1 thru 9, transmit
                                                  ; data bits (8 total)

;
TXM_STOP_0: SETB P3.2                            ; When bit count = 0,
                                                  ; transmit stop bit
          CLR TXM_IN_PROGRESS_0                ; Indicate transmission is
                                                  ; finished and ready for
                                                  ; next byte
          JMP TXM_EXIT

;
TXM_DATA_0: MOV A, TXM_REG_0                    ; Transmit one bit at a time
          RRC A                                    ; through the carry bit
          MOV P3.2, C
          MOV TXM_REG_0, A                      ; Save what's not been sent

;
TXM_EXIT: CLR C                                  ; Update compare value with
          MOV A, #FULL_BIT_LOW                 ; Full bit time = 22CH
          ADD A, CCAP3L
          MOV CCAP3L, A
          MOV A, #FULL_BIT_HIGH
          ADDC A, CCAP3H
          MOV CCAP3H, A
          POP PSW
          POP ACC
          RETI

```

270531-10

Conclusion

The software routines in the Appendix can be altered to vary the baud rate and number of channels to fit a particular application. The number of channels which can be implemented is limited by the CPU time required to service the PCA interrupt. At higher baud rates, fewer channels can be run.

The test program verifies the simultaneous operation of three half-duplex channels at 2400 Baud and the on-chip full-duplex channel at 9600 Baud. Thirty-three percent of the CPU time is required to operate all four channels. The test was run for several hours with no apparent malfunctions.

APPENDIX

MCS-51 MACRO ASSEMBLER SWPORT

01/01/80 PAGE 1

DOS 3.20 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN SWPORT.OBJ
ASSEMBLER INVOKED BY: C:\AEDIT\ASM51.EXE SWPORT.RCV

```

LOC OBJ      LINE      SOURCE
0000 0000      1      $NOMOD51
0000 0000      2      $NOSYMBOLS
0000 0000      3      $NOLIST
0000 0000     152      ;
0000 0000     153      ;
0000 0000     154      ;
0000 0000     155      ; This program tests the receive routines of a software serial port.
0000 0000     156      ; Three half-duplex channels are implemented in software to run at
0000 0000     157      ; 2400 Baud (16MHz). The on-chip serial port is also running full-duplex
0000 0000     158      ; at 9600 Baud. Thirty-three percent of the CPU time is required to run
0000 0000     159      ; all four ports simultaneously.
0000 0000     160      ;
0000 0000     161      ; To test the receive routines, "dummy" terminals transmit 00 - FF hex
0000 0000     162      ; continually to the PCA. When the first byte is received, it is
0000 0000     163      ; compared with 00. If the comparison is valid, the compare value is
0000 0000     164      ; incremented and the routine waits to receive the next byte. Error
0000 0000     165      ; routines toggle various Port 3 pins if an invalid comparison occurs
0000 0000     166      ; or if an invalid start bit or stop bit is received.
0000 0000     167      ;
0000 0000     168      ORG 00H
0000 020036    169      LJMPL INITIALIZE
0000 0000     170      ;
001B 0000     171      ORG 001BH
001B 02025C    172      LJMPL RECEIVE_DONE ; Timer 1 Overflow - simulates "RI" interrupt
0023 0000     173      ;
0023 020282    174      ORG 0023H
0023 0000     175      LJMPL SERIAL_PORT ; Serial port interrupt
0033 0000     176      ;
0033 0200DC    177      ORG 0033H ; PCA interrupt
0033 0000     178      LJMPL RECEIVE
0000 0000     179      ;
0000 0000     180      ;
0000 0000     181      ;
0000 0000     182      ;
0000 0000     183      ;
0000 0000     184      ;
0000 0000     185      ;
0000 0000     186      ;
0000 0000     187      ;
0000 0000     188      ;
0000 0000     189      ;
0000 0000     190      ;
0000 0000     191      ;
0000 0000     192      ;
0000 0000     193      ;
0000 0000     194      ;
0000 0000     195      ;
0000 0000     196      ;
0000 0000     197      ;
0000 0000     198      ;

```

VARIABLES USED BY THE SOFTWARE SERIAL PORT

RECEIVE ROUTINE:

```

0000 0000     187      RCV_START_BIT_0      BIT      20H.0 ; Indicates start bit has been
0000 0000     188      RCV_START_BIT_1      BIT      21H.0 ; received
0000 0000     189      RCV_START_BIT_2      BIT      22H.0
0000 0000     190      ;
0000 0000     191      RCV_DONE_0          BIT      20H.1 ; Indicates data byte has been
0000 0000     192      RCV_DONE_1          BIT      21H.1 ; received
0000 0000     193      RCV_DONE_2          BIT      22H.1
0000 0000     194      ;
0000 0000     195      RCV_ON_0          BIT      20H.2 ; Used in main test program to check
0000 0000     196      RCV_ON_1          BIT      21H.2 ; for a received byte
0000 0000     197      RCV_ON_2          BIT      22H.2
0000 0000     198      ;

```

270531-11

LOC	OBJ	LINE	SOURCE
0030		199	RCV_BUF_0 DATA 30H ; Software receive "SBUF"
0040		200	RCV_BUF_1 DATA 40H
0050		201	RCV_BUF_2 DATA 50H
		202	;
0031		203	RCV_REG_0 DATA 31H ; Temporary register for
0041		204	RCV_REG_1 DATA 41H ; receiving bits
0051		205	RCV_REG_2 DATA 51H
		206	;
0032		207	RCV_COUNT_0 DATA 32H ; Counter for receiving bits
0042		208	RCV_COUNT_1 DATA 42H
0052		209	RCV_COUNT_2 DATA 52H
		210	;
0033		211	COUNT_0 DATA 33H ; Used in test program to check
0043		212	COUNT_1 DATA 43H ; bytes being received
0053		213	COUNT_2 DATA 53H
		214	;
0011		215	NEG_EDGE EQU 11H ; Two modes of operation for the
0049		216	S_W_TIMER EQU 49H ; Compare/Capture modules
		217	;
0015		218	HALF_BIT_LOW EQU 15H ; Half bit time = 115H
0001		219	HALF_BIT_HIGH EQU 01H
002C		220	FULL_BIT_LOW EQU 2CH ; Full bit time = 22CH
0002		221	FULL_BIT_HIGH EQU 02H ; 2400 Baud @ 16MHz
		222	;
		223	;
		224	;
		225	;
		226	;
		227	;
		228	;
		229	;
0036 75815F		230	INITIALIZE: MOV SP, #5FH ; Initialize stack pointer
		231	;
0039 75D900		232	INIT_PCA: MOV CMOD, #00H ; (specific to the test program)
003C 75D800		233	MOV CCON, #00H ; Increment PCA clock @ 1/12 Osc Freq
003F 75DA11		234	MOV CCAPM0, #NEG_EDGE ; Clear all status flags
0042 75DB11		235	MOV CCAPM1, #NEG_EDGE ; Module 0 in Neg-edge capture mode (Pl.3)
0045 75DC11		236	MOV CCAPM2, #NEG_EDGE ; Module 1 (Pl.4)
		237	;
0048 75E900		238	MOV CL, #00H ; Module 2 (Pl.5)
004B 75F900		239	MOV CH, #00H
004E 75A8D8		240	MOV IE, #0D8H ; Initialize needed interrupt: EA,EC,ES,ET1
0051 D2DE		241	SETB CR ; Turn on PCA counter
		242	;
0053 759850		243	INIT_SP: MOV SCON, #50H ; Serial port in mode 1 (8-Bit UART)
0056 75CBFF		244	MOV RCAP2H, #0FFH ; Reload values for 9600 Baud @ 16 MHz
0059 75CACC		245	MOV RCAP2L, #0CCH
005C 75C834		246	MOV T2CON, #34H ; Timer 2 as a baud-rate generator,
		247	; turn on timer 2
		248	;
005F C200		249	INIT_FLAGS: CLR RCV_START_BIT_0
0061 C208		250	CLR RCV_START_BIT_1
0063 C210		251	CLR RCV_START_BIT_2
		252	;
0065 C201		253	CLR RCV_DONE_0

LOC	OBJ	LINE	SOURCE
0067	C209	254	CLR RCV_DONE_1
0069	C211	255	CLR RCV_DONE_2
		256	
006B	C202	257	CLR RCV_ON 0
006D	C20A	258	CLR RCV_ON_1
006F	C212	259	CLR RCV_ON_2
		260	
		261	; Port 3 pins used in test program for error routines
		262	
		263	; Main program:
0071	D2B2	264	SETB P3.2 ; Error in comparison on module 0
0073	D2B3	265	SETB P3.3 ; Error in comparison on module 1
0075	D2B4	266	SETB P3.4 ; Error in comparison on module 2
		267	
		268	; Interrupt routines:
0077	D2B5	269	SETB P3.5 ; Error in reception on module 0
0079	D2B6	270	SETB P3.6 ; Error in reception on module 1
007B	D2B7	271	SETB P3.7 ; Error in reception on module 2
		272	
007D	753000	273	MOV RCV_BUF 0, #00H
0080	754000	274	MOV RCV_BUF_1, #00H
0083	755000	275	MOV RCV_BUF_2, #00H
		276	
0086	753200	277	MOV RCV_COUNT 0, #00H
0089	754200	278	MOV RCV_COUNT_1, #00H
008C	755200	279	MOV RCV_COUNT_2, #00H
		280	
008F	753100	281	MOV RCV_REG 0, #00H
0092	754100	282	MOV RCV_REG_1, #00H
0095	755100	283	MOV RCV_REG_2, #00H
		284	
0098	753300	285	MOV COUNT 0, #00H
009B	754300	286	MOV COUNT_1, #00H
009E	755300	287	MOV COUNT_2, #00H
		288	
		289	
		290	
		291	
		292	
		293	
		294	
00A1	300209	294	CHECK_0: JNB RCV_ON 0, CHECK_1 ; Main program continually checks
00A4	E530	295	MOV A, RCV_BUF 0 ; each channel for a received byte.
00A6	B5331E	296	CJNE A, COUNT_0, ERROR0 ; Once a byte is received, it is compared
00A9	C202	297	CLR RCV_ON 0 ; with the current value in the "COUNT"
00AB	0533	298	INC COUNT_0 ; register
		299	
00AD	300A09	300	CHECK_1: JNB RCV_ON 1, CHECK_2
00B0	E540	301	MOV A, RCV_BUF 1
00B2	B54319	302	CJNE A, COUNT_1, ERROR1
00B5	C20A	303	CLR RCV_ON 1
00B7	0543	304	INC COUNT_1
		305	
00B9	3012E5	306	CHECK_2: JNB RCV_ON 2, CHECK_0
00BC	E550	307	MOV A, RCV_BUF_2
00BE	B55314	308	CJNE A, COUNT_2, ERROR2

LOC	OBJ	LINE	SOURCE
00C1	C212	309	CLR RCV ON 2
00C3	0553	310	INC COUNT 2
00C5	80DA	311	JMP CHECK_0
00C7	C2B2	312	;
00C9	75DA00	313	ERROR0: CLR P3.2 ; Error in comparison on module 0
00CC	80DF	314	MOV CCAPM0, #00H ; Discontinue receiving bytes
00CE	C2B3	315	JMP CHECK_1
00D0	75DB00	316	;
00D3	80E4	317	ERROR1: CLR P3.3 ; Error in comparison on module 1
00D5	C2B4	318	MOV CCAPM1, #00H
00D7	75DC00	319	JMP CHECK_2
00DA	80C5	320	;
00DB		321	ERROR2: CLR P3.4 ; Error in comparison on module 2
00DC	C0E0	322	MOV CCAPM2, #00H
00DE	C0D0	323	JMP CHECK_0
00E0	20D811	324	;
00E3	20D908	325	PCA INTERRUPT ROUTINE - RECEIVE BITS
00E6	20DA08	326	=====
00E9	D0D0	327	RECEIVE: PUSH ACC
00EB	D0E0	328	PUSH PSW
00ED	32	329	;
00EE	02016C	330	JB CCF0, MODULE_0 ; Check which module caused
00F1	0201E4	331	JB CCF1, JUMP_1 ; PCA interrupt and jump to
00F3		332	JB CCF2, JUMP_2 ; appropriate routine
00F5		333	POP PSW
00F7		334	POP ACC
00F9		335	RETI
00FB		336	;
00FD	C3	337	JUMP_1: LJMP MODULE_1
00FF	7415	338	JUMP_2: LJMP MODULE_2
0100	25EA	339	;
0102	F5EA	340	;
0104	7401	341	;
0106	35FA	342	;
0108	F5FA	343	;
010A	75DA49	344	;
010D	D0D0	345	;
010F	D0E0	346	;
0111		347	;
0113		348	;
0115		349	;
0117		350	;
0119		351	;
011B		352	;
011D		353	;
011F		354	;
0121		355	;
0123		356	;
0125		357	;
0127		358	;
0129		359	;
012B		360	;
012D		361	;
012F		362	;
0131		363	;
0133		364	;
0135		365	;
0137		366	;
0139		367	;
013B		368	;
013D		369	;
013F		370	;
0141		371	;
0143		372	;
0145		373	;
0147		374	;
0149		375	;
014B		376	;
014D		377	;
014F		378	;
0151		379	;
0153		380	;
0155		381	;
0157		382	;
0159		383	;
015B		384	;
015D		385	;
015F		386	;
0161		387	;
0163		388	;
0165		389	;
0167		390	;
0169		391	;
016B		392	;
016D		393	;
016F		394	;
0171		395	;
0173		396	;
0175		397	;
0177		398	;
0179		399	;
017B		400	;
017D		401	;
017F		402	;
0181		403	;
0183		404	;
0185		405	;
0187		406	;
0189		407	;
018B		408	;
018D		409	;
018F		410	;
0191		411	;
0193		412	;
0195		413	;
0197		414	;
0199		415	;
019B		416	;
019D		417	;
019F		418	;
01A1		419	;
01A3		420	;
01A5		421	;
01A7		422	;
01A9		423	;
01AB		424	;
01AD		425	;
01AF		426	;
01B1		427	;
01B3		428	;
01B5		429	;
01B7		430	;
01B9		431	;
01BB		432	;
01BD		433	;
01BF		434	;
01C1		435	;
01C3		436	;
01C5		437	;
01C7		438	;
01C9		439	;
01CB		440	;
01CD		441	;
01CF		442	;
01D1		443	;
01D3		444	;
01D5		445	;
01D7		446	;
01D9		447	;
01DB		448	;
01DD		449	;
01DF		450	;
01E1		451	;
01E3		452	;
01E5		453	;
01E7		454	;
01E9		455	;
01EB		456	;
01ED		457	;
01EF		458	;
01F1		459	;
01F3		460	;
01F5		461	;
01F7		462	;
01F9		463	;
01FB		464	;
01FD		465	;
01FF		466	;

270531-14

intel

AB-41

10711


```

LOC OBJ      LINE      SOURCE
0111 32       364       RETI
0112 B4494B   365       ;
0112 B4494B   366       RCV_START_0: CJNE A, #S_W_TIMER, ERROR_0 ; Check module is configured
0115 20001A   367       ; as a software timer, otherwise error.
0115 20001A   368       JB RCV_START_BIT_0, RCV_BYTE_0 ; Check if start bit
0118 209345   369       ; has been received yet
0118 209345   370       JB P1.3, ERROR_0 ; Check that start bit = 0,
0118 209345   371       ; otherwise error.
011B D200     372       SETB RCV_START_BIT_0 ; Signify valid start bit
011D 753209   373       ; was received
011D 753209   374       MOV RCV_COUNT_0, #09H ; Start counting bits sampled
0120 C3       375       ;
0120 C3       376       CLR C ; Update C/C registers to sample
0121 742C     377       MOV A, #FULL_BIT_LOW ; incoming bits
0123 25EA     378       ADD A, CCAPOL ; Full bit time = 22CH
0125 F5EA     379       MOV CCAPOL, A
0127 7402     380       MOV A, #FULL_BIT_HIGH
0129 35FA     381       ADDC A, CCAPUH
012B F5FA     382       MOV CCAPUH, A
012D D0D0     383       POP PSW
012F D0E0     384       POP ACC
0131 32       385       RETI
0132 D53212   386       ;
0132 D53212   387       RCV_BYTE_0: DJNZ RCV_COUNT_0, RCV_DATA_0 ; On 9th sample, check for
0135 309328   388       ; valid stop bit
0135 309328   389       RCV_STOP_0: JNB P1.3, ERROR_0
0138 853130   390       MOV RCV_BUF_0, RCV_REG_0 ; Save received byte in receive "SBUF"
0138 853130   391       SETB RCV_DONE_0 ; Flag which module received a byte
013B D201     392       SETB TFI ; Generate an interrupt so main program
013D D28F     393       ; knows a byte has been received
013F 75DA11   394       ; (NOTE: selection of TFI is arbitrary)
013F 75DA11   395       MOV CCAPM0, #NEG_EDGE ; Reconfigure module 0 for next
0142 D0D0     396       POP PSW ; reception of a start bit
0144 D0E0     397       POP ACC
0146 32       398       RETI
0147 A293     399       ;
0147 A293     400       RCV_DATA_0: MOV C, P1.3 ; Sampling data bits
0149 E531     401       MOV A, RCV_REG_0 ; Shift bits through CY into ACC
014B 13       402       RRC A
014C F531     403       MOV RCV_REG_0, A ; Save each reception in temporary
014E C3       404       ; register
014E C3       405       CLR C
014F 742C     406       MOV A, #FULL_BIT_LOW ; Update C/C register for next
0151 25EA     407       ADD A, CCAPOL ; sample time
0153 F5EA     408       MOV CCAPOL, A
0155 7402     409       MOV A, #FULL_BIT_HIGH
0157 35FA     410       ADDC A, CCAPUH
0159 F5FA     411       MOV CCAPUH, A
015B D0D0     412       POP PSW
015D D0E0     413       POP ACC
015F 32       414       RETI
0160 C2B5     415       ;
0160 C2B5     416       ERROR_0: CLR P3.5 ; Error routine for invalid start or
0160 C2B5     417       ; stop bit or invalid mode comparison
0162 32       418       RETI
0162 32       419       ;
0162 32       420       ;
0162 32       421       ;
0162 32       422       ;
0162 32       423       ;
0162 32       424       ;
0162 32       425       ;
0162 32       426       ;
0162 32       427       ;
0162 32       428       ;
0162 32       429       ;
0162 32       430       ;
0162 32       431       ;
0162 32       432       ;
0162 32       433       ;
0162 32       434       ;
0162 32       435       ;
0162 32       436       ;
0162 32       437       ;
0162 32       438       ;
0162 32       439       ;
0162 32       440       ;
0162 32       441       ;
0162 32       442       ;
0162 32       443       ;
0162 32       444       ;
0162 32       445       ;
0162 32       446       ;
0162 32       447       ;
0162 32       448       ;
0162 32       449       ;
0162 32       450       ;
0162 32       451       ;
0162 32       452       ;
0162 32       453       ;
0162 32       454       ;
0162 32       455       ;
0162 32       456       ;
0162 32       457       ;
0162 32       458       ;
0162 32       459       ;
0162 32       460       ;
0162 32       461       ;
0162 32       462       ;
0162 32       463       ;
0162 32       464       ;
0162 32       465       ;
0162 32       466       ;
0162 32       467       ;
0162 32       468       ;
0162 32       469       ;
0162 32       470       ;
0162 32       471       ;
0162 32       472       ;
0162 32       473       ;
0162 32       474       ;
0162 32       475       ;
0162 32       476       ;
0162 32       477       ;
0162 32       478       ;
0162 32       479       ;
0162 32       480       ;
0162 32       481       ;
0162 32       482       ;
0162 32       483       ;
0162 32       484       ;
0162 32       485       ;
0162 32       486       ;
0162 32       487       ;
0162 32       488       ;
0162 32       489       ;
0162 32       490       ;
0162 32       491       ;
0162 32       492       ;
0162 32       493       ;
0162 32       494       ;
0162 32       495       ;
0162 32       496       ;
0162 32       497       ;
0162 32       498       ;
0162 32       499       ;
0162 32       500       ;
0162 32       501       ;
0162 32       502       ;
0162 32       503       ;
0162 32       504       ;
0162 32       505       ;
0162 32       506       ;
0162 32       507       ;
0162 32       508       ;
0162 32       509       ;
0162 32       510       ;
0162 32       511       ;
0162 32       512       ;
0162 32       513       ;
0162 32       514       ;
0162 32       515       ;
0162 32       516       ;
0162 32       517       ;
0162 32       518       ;
0162 32       519       ;
0162 32       520       ;
0162 32       521       ;
0162 32       522       ;
0162 32       523       ;
0162 32       524       ;
0162 32       525       ;
0162 32       526       ;
0162 32       527       ;
0162 32       528       ;
0162 32       529       ;
0162 32       530       ;
0162 32       531       ;
0162 32       532       ;
0162 32       533       ;
0162 32       534       ;
0162 32       535       ;
0162 32       536       ;
0162 32       537       ;
0162 32       538       ;
0162 32       539       ;
0162 32       540       ;
0162 32       541       ;
0162 32       542       ;
0162 32       543       ;
0162 32       544       ;
0162 32       545       ;
0162 32       546       ;
0162 32       547       ;
0162 32       548       ;
0162 32       549       ;
0162 32       550       ;
0162 32       551       ;
0162 32       552       ;
0162 32       553       ;
0162 32       554       ;
0162 32       555       ;
0162 32       556       ;
0162 32       557       ;
0162 32       558       ;
0162 32       559       ;
0162 32       560       ;
0162 32       561       ;
0162 32       562       ;
0162 32       563       ;
0162 32       564       ;
0162 32       565       ;
0162 32       566       ;
0162 32       567       ;
0162 32       568       ;
0162 32       569       ;
0162 32       570       ;
0162 32       571       ;
0162 32       572       ;
0162 32       573       ;
0162 32       574       ;
0162 32       575       ;
0162 32       576       ;
0162 32       577       ;
0162 32       578       ;
0162 32       579       ;
0162 32       580       ;
0162 32       581       ;
0162 32       582       ;
0162 32       583       ;
0162 32       584       ;
0162 32       585       ;
0162 32       586       ;
0162 32       587       ;
0162 32       588       ;
0162 32       589       ;
0162 32       590       ;
0162 32       591       ;
0162 32       592       ;
0162 32       593       ;
0162 32       594       ;
0162 32       595       ;
0162 32       596       ;
0162 32       597       ;
0162 32       598       ;
0162 32       599       ;
0162 32       600       ;
0162 32       601       ;
0162 32       602       ;
0162 32       603       ;
0162 32       604       ;
0162 32       605       ;
0162 32       606       ;
0162 32       607       ;
0162 32       608       ;
0162 32       609       ;
0162 32       610       ;
0162 32       611       ;
0162 32       612       ;
0162 32       613       ;
0162 32       614       ;
0162 32       615       ;
0162 32       616       ;
0162 32       617       ;
0162 32       618       ;
0162 32       619       ;
0162 32       620       ;
0162 32       621       ;
0162 32       622       ;
0162 32       623       ;
0162 32       624       ;
0162 32       625       ;
0162 32       626       ;
0162 32       627       ;
0162 32       628       ;
0162 32       629       ;
0162 32       630       ;
0162 32       631       ;
0162 32       632       ;
0162 32       633       ;
0162 32       634       ;
0162 32       635       ;
0162 32       636       ;
0162 32       637       ;
0162 32       638       ;
0162 32       639       ;
0162 32       640       ;
0162 32       641       ;
0162 32       642       ;
0162 32       643       ;
0162 32       644       ;
0162 32       645       ;
0162 32       646       ;
0162 32       647       ;
0162 32       648       ;
0162 32       649       ;
0162 32       650       ;
0162 32       651       ;
0162 32       652       ;
0162 32       653       ;
0162 32       654       ;
0162 32       655       ;
0162 32       656       ;
0162 32       657       ;
0162 32       658       ;
0162 32       659       ;
0162 32       660       ;
0162 32       661       ;
0162 32       662       ;
0162 32       663       ;
0162 32       664       ;
0162 32       665       ;
0162 32       666       ;
0162 32       667       ;
0162 32       668       ;
0162 32       669       ;
0162 32       670       ;
0162 32       671       ;
0162 32       672       ;
0162 32       673       ;
0162 32       674       ;
0162 32       675       ;
0162 32       676       ;
0162 32       677       ;
0162 32       678       ;
0162 32       679       ;
0162 32       680       ;
0162 32       681       ;
0162 32       682       ;
0162 32       683       ;
0162 32       684       ;
0162 32       685       ;
0162 32       686       ;
0162 32       687       ;
0162 32       688       ;
0162 32       689       ;
0162 32       690       ;
0162 32       691       ;
0162 32       692       ;
0162 32       693       ;
0162 32       694       ;
0162 32       695       ;
0162 32       696       ;
0162 32       697       ;
0162 32       698       ;
0162 32       699       ;
0162 32       700       ;
0162 32       701       ;
0162 32       702       ;
0162 32       703       ;
0162 32       704       ;
0162 32       705       ;
0162 32       706       ;
0162 32       707       ;
0162 32       708       ;
0162 32       709       ;
0162 32       710       ;
0162 32       711       ;
0162 32       712       ;
0162 32       713       ;
0162 32       714       ;
0162 32       715       ;
0162 32       716       ;
0162 32       717       ;
0162 32       718       ;
0162 32       719       ;
0162 32       720       ;
0162 32       721       ;
0162 32       722       ;
0162 32       723       ;
0162 32       724       ;
0162 32       725       ;
0162 32       726       ;
0162 32       727       ;
0162 32       728       ;
0162 32       729       ;
0162 32       730       ;
0162 32       731       ;
0162 32       732       ;
0162 32       733       ;
0162 32       734       ;
0162 32       735       ;
0162 32       736       ;
0162 32       737       ;
0162 32       738       ;
0162 32       739       ;
0162 32       740       ;
0162 32       741       ;
0162 32       742       ;
0162 32       743       ;
0162 32       744       ;
0162 32       745       ;
0162 32       746       ;
0162 32       747       ;
0162 32       748       ;
0162 32       749       ;
0162 32       750       ;
0162 32       751       ;
0162 32       752       ;
0162 32       753       ;
0162 32       754       ;
0162 32       755       ;
0162 32       756       ;
0162 32       757       ;
0162 32       758       ;
0162 32       759       ;
0162 32       760       ;
0162 32       761       ;
0162 32       762       ;
0162 32       763       ;
0162 32       764       ;
0162 32       765       ;
0162 32       766       ;
0162 32       767       ;
0162 32       768       ;
0162 32       769       ;
0162 32       770       ;
0162 32       771       ;
0162 32       772       ;
0162 32       773       ;
0162 32       774       ;
0162 32       775       ;
0162 32       776       ;
0162 32       777       ;
0162 32       778       ;
0162 32       779       ;
0162 32       780       ;
0162 32       781       ;
0162 32       782       ;
0162 32       783       ;
0162 32       784       ;
0162 32       785       ;
0162 32       786       ;
0162 32       787       ;
0162 32       788       ;
0162 32       789       ;
0162 32       790       ;
0162 32       791       ;
0162 32       792       ;
0162 32       793       ;
0162 32       794       ;
0162 32       795       ;
0162 32       796       ;
0162 32       797       ;
0162 32       798       ;
0162 32       799       ;
0162 32       800       ;
0162 32       801       ;
0162 32       802       ;
0162 32       803       ;
0162 32       804       ;
0162 32       805       ;
0162 32       806       ;
0162 32       807       ;
0162 32       808       ;
0162 32       809       ;
0162 32       810       ;
0162 32       811       ;
0162 32       812       ;
0162 32       813       ;
0162 32       814       ;
0162 32       815       ;
0162 32       816       ;
0162 32       817       ;
0162 32       818       ;
0162 32       819       ;
0162 32       820       ;
0162 32       821       ;
0162 32       822       ;
0162 32       823       ;
0162 32       824       ;
0162 32       825       ;
0162 32       826       ;
0162 32       827       ;
0162 32       828       ;
0162 32       829       ;
0162 32       830       ;
0162 32       831       ;
0162 32       832       ;
0162 32       833       ;
0162 32       834       ;
0162 32       835       ;
0162 32       836       ;
0162 32       837       ;
0162 32       838       ;
0162 32       839       ;
0162 32       840       ;
0162 32       841       ;
0162 32       842       ;
0162 32       843       ;
0162 32       844       ;
0162 32       845       ;
0162 32       846       ;
0162 32       847       ;
0162 32       848       ;
0162 32       849       ;
0162 32       850       ;
0162 32       851       ;
0162 32       852       ;
0162 32       853       ;
0162 32       854       ;
0162 32       855       ;
0162 32       856       ;
0162 32       857       ;
0162 32       858       ;
0162 32       859       ;
0162 32       860       ;
0162 32       861       ;
0162 32       862       ;
0162 32       863       ;
0162 32       864       ;
0162 32       865       ;
0162 32       866       ;
0162 32       867       ;
0162 32       868       ;
0162 32       869       ;
0162 32       870       ;
0162 32       871       ;
0162 32       872       ;
0162 32       873       ;
0162 32       874       ;
0162 32       875       ;
0162 32       876       ;
0162 32       877       ;
0162 32       878       ;
0162 32       879       ;
0162 32       880       ;
0162 32       881       ;
0162 32       882       ;
0162 32       883       ;
0162 32       884       ;
0162 32       885       ;
0162 32       886       ;
0162 32       887       ;
0162 32       888       ;
0162 32       889       ;
0162 32       890       ;
0162 32       891       ;
0162 32       892       ;
0162 32       893       ;
0162 32       894       ;
0162 32       895       ;
0162 32       896       ;
0162 32       897       ;
0162 32       898       ;
0162 32       899       ;
0162 32       900       ;
0162 32       901       ;
0162 32       902       ;
0162 32       903       ;
0162 32       904       ;
0162 32       905       ;
0162 32       906       ;
0162 32       907       ;
0162 32       908       ;
0162 32       909       ;
0162 32       910       ;
0162 32       911       ;
0162 32       912       ;
0162 32       913       ;
0162 32       914       ;
0162 32       915       ;
0162 32       916       ;
0162 32       917       ;
0162 32       918       ;
0162 32       919       ;
0162 32       920       ;
0162 32       921       ;
0162 32       922       ;
0162 32       923       ;
0162 32       924       ;
0162 32       925       ;
0162 32       926       ;
0162 32       927       ;
0162 32       928       ;
0162 32       929       ;
0162 32       930       ;
0162 32       931       ;
0162 32       932       ;
0162 32       933       ;
0162 32       934       ;
0162 32       935       ;
0162 32       936       ;
0162 32       937       ;
0162 32       938       ;
0162 32       939       ;
0162 32       940       ;
0162 32       941       ;
0162 32       942       ;
0162 32       943       ;
0162 32       944       ;
0162 32       945       ;
0162 32       946       ;
0162 32       947       ;
0162 32       948       ;
0162 32       949       ;
0162 32       950       ;
0162 32       951       ;
0162 32       952       ;
0162 32       953       ;
0162 32       954       ;
0162 32       955       ;
0162 32       956       ;
0162 32       957       ;
0162 32       958       ;
0162 32       959       ;
0162 32       960       ;
0162 32       961       ;
0162 32       962       ;
0162 32       963       ;
0162 32       964       ;
0162 32       965       ;
0162 32       966       ;
0162 32       967       ;
0162 32       968       ;
0162 32       969       ;
0162 32       970       ;
0162 32       971       ;
0162 32       972       ;
0162 32       973       ;
0162 32       974       ;
0162 32       975       ;
0162 32       976       ;
0162 32       977       ;
0162 32       978       ;
0162 32       979       ;
0162 32       980       ;
0162 32       981       ;
0162 32       982       ;
0162 32       983       ;
0162 32       984       ;
0162 32       985       ;
0162 32       986       ;
0162 32       987       ;
0162 32       988       ;
0162 32       989       ;
0162 32       990       ;
0162 32       991       ;
0162 32       992       ;
0162 32       993       ;
0162 32       994       ;
0162 32       995       ;
0162 32       996       ;
0162 32       997       ;
0162 32       998       ;
0162 32       999       ;
0162 32      1000       ;

```

270531-15

intel

AB-41

16th

2-236

270531-16

```

LOC OBJ      LINE  SOURCE
01BE 32      473      RETI
01BF A294    474      ;
01C1 E541    475      RCV_DATA_1:  MOV C, P1.4
01C3 13      476      MOV A, RCV_REG_1
01C4 F541    477      RRC A
01C5         478      MOV RCV_REG_1, A
01C6 C3      479      ;
01C7 742C    480      CLR C
01C9 25EB    481      MOV A, #FULL BIT_LOW
01CB F5EB    482      ADD A, CCAP1L
01CD 7402    483      MOV CCAP1L, A
01CF 35FB    484      MOV A, #FULL BIT_HIGH
01D1 F5FB    485      ADDC A, CCAP1H
01D3 D0D0    486      MOV CCAP1H, A
01D5 D0E0    487      POP PSW
01D7 32      488      POP ACC
01D8 C2B6    489      RETI
01DA 75DB11  490      ;
01DD C208    491      ERROR_1:  CLR P3.6
01DF D0D0    492      MOV CCAPM1, #NEG_EDGE
01E1 D0E0    493      CLR RCV_START_BIT_1
01E3 32      494      POP PSW
01E4 C2DA    495      POP ACC
01E5 E5DC    496      RETI
01E8 547F    497      ;
01EA B41115  498      ;
01ED C3      499      ;
01EE 7415    500      ; CHANNEL 2
01F0 25EC    501      ;
01F2 F5EC    502      ;
01F4 7401    503      ;
01F6 35FC    504      MODULE_2: CLR CCF2 ; Similar to module 0
01F8 F5FC    505      MOV A, CCAPM2
01FA 75DC49  506      ANL A, #01111111B
01FC D0D0    507      CJNE A, #NEG_EDGE, RCV_START_2
01FE D0E0    508      ;
0201 32      509      CLR C
0202 B4494B  510      MOV A, #HALF BIT_LOW
0205 20101A  511      ADD A, CCAP2L
0208 209545  512      MOV CCAP2L, A
020B D210    513      MOV A, #HALF BIT_HIGH
020D 755209  514      ADDC A, CCAP2H
0210         515      MOV CCAP2H, A
0211         516      MOV CCAPM2, #S_W_TIMER
0212         517      POP PSW
0213         518      POP ACC
0214         519      RETI
0215         520      ;
0216         521      RCV_START_2: CJNE A, #S_W_TIMER, ERROR_2
0217         522      JB RCV_START_BIT_2, RCV_BYTE_2
0218         523      JB P1.5, ERROR_2
0219         524      ;
0220         525      SETB RCV_START_BIT_2
0221         526      MOV RCV_COUNT_2, #09H
0222         527      ;
0223         528      ;
0224         529      ;
0225         530      ;
0226         531      ;
0227         532      ;
0228         533      ;
0229         534      ;
0230         535      ;
0231         536      ;
0232         537      ;
0233         538      ;
0234         539      ;
0235         540      ;
0236         541      ;
0237         542      ;
0238         543      ;
0239         544      ;
0240         545      ;
0241         546      ;
0242         547      ;
0243         548      ;
0244         549      ;
0245         550      ;
0246         551      ;
0247         552      ;
0248         553      ;
0249         554      ;
0250         555      ;
0251         556      ;
0252         557      ;
0253         558      ;
0254         559      ;
0255         560      ;
0256         561      ;
0257         562      ;
0258         563      ;
0259         564      ;
0260         565      ;
0261         566      ;
0262         567      ;
0263         568      ;
0264         569      ;
0265         570      ;
0266         571      ;
0267         572      ;
0268         573      ;
0269         574      ;
0270         575      ;
0271         576      ;
0272         577      ;
0273         578      ;
0274         579      ;
0275         580      ;
0276         581      ;
0277         582      ;
0278         583      ;
0279         584      ;
0280         585      ;
0281         586      ;
0282         587      ;
0283         588      ;
0284         589      ;
0285         590      ;
0286         591      ;
0287         592      ;
0288         593      ;
0289         594      ;
0290         595      ;
0291         596      ;
0292         597      ;
0293         598      ;
0294         599      ;
0295         600      ;
0296         601      ;
0297         602      ;
0298         603      ;
0299         604      ;
0300         605      ;
0301         606      ;
0302         607      ;
0303         608      ;
0304         609      ;
0305         610      ;
0306         611      ;
0307         612      ;
0308         613      ;
0309         614      ;
0310         615      ;
0311         616      ;
0312         617      ;
0313         618      ;
0314         619      ;
0315         620      ;
0316         621      ;
0317         622      ;
0318         623      ;
0319         624      ;
0320         625      ;
0321         626      ;
0322         627      ;
0323         628      ;
0324         629      ;
0325         630      ;
0326         631      ;
0327         632      ;
0328         633      ;
0329         634      ;
0330         635      ;
0331         636      ;
0332         637      ;
0333         638      ;
0334         639      ;
0335         640      ;
0336         641      ;
0337         642      ;
0338         643      ;
0339         644      ;
0340         645      ;
0341         646      ;
0342         647      ;
0343         648      ;
0344         649      ;
0345         650      ;
0346         651      ;
0347         652      ;
0348         653      ;
0349         654      ;
0350         655      ;
0351         656      ;
0352         657      ;
0353         658      ;
0354         659      ;
0355         660      ;
0356         661      ;
0357         662      ;
0358         663      ;
0359         664      ;
0360         665      ;
0361         666      ;
0362         667      ;
0363         668      ;
0364         669      ;
0365         670      ;
0366         671      ;
0367         672      ;
0368         673      ;
0369         674      ;
0370         675      ;
0371         676      ;
0372         677      ;
0373         678      ;
0374         679      ;
0375         680      ;
0376         681      ;
0377         682      ;
0378         683      ;
0379         684      ;
0380         685      ;
0381         686      ;
0382         687      ;
0383         688      ;
0384         689      ;
0385         690      ;
0386         691      ;
0387         692      ;
0388         693      ;
0389         694      ;
0390         695      ;
0391         696      ;
0392         697      ;
0393         698      ;
0394         699      ;
0395         700      ;
0396         701      ;
0397         702      ;
0398         703      ;
0399         704      ;
0400         705      ;
0401         706      ;
0402         707      ;
0403         708      ;
0404         709      ;
0405         710      ;
0406         711      ;
0407         712      ;
0408         713      ;
0409         714      ;
0410         715      ;
0411         716      ;
0412         717      ;
0413         718      ;
0414         719      ;
0415         720      ;
0416         721      ;
0417         722      ;
0418         723      ;
0419         724      ;
0420         725      ;
0421         726      ;
0422         727      ;
0423         728      ;
0424         729      ;
0425         730      ;
0426         731      ;
0427         732      ;
0428         733      ;
0429         734      ;
0430         735      ;
0431         736      ;
0432         737      ;
0433         738      ;
0434         739      ;
0435         740      ;
0436         741      ;
0437         742      ;
0438         743      ;
0439         744      ;
0440         745      ;
0441         746      ;
0442         747      ;
0443         748      ;
0444         749      ;
0445         750      ;
0446         751      ;
0447         752      ;
0448         753      ;
0449         754      ;
0450         755      ;
0451         756      ;
0452         757      ;
0453         758      ;
0454         759      ;
0455         760      ;
0456         761      ;
0457         762      ;
0458         763      ;
0459         764      ;
0460         765      ;
0461         766      ;
0462         767      ;
0463         768      ;
0464         769      ;
0465         770      ;
0466         771      ;
0467         772      ;
0468         773      ;
0469         774      ;
0470         775      ;
0471         776      ;
0472         777      ;
0473         778      ;
0474         779      ;
0475         780      ;
0476         781      ;
0477         782      ;
0478         783      ;
0479         784      ;
0480         785      ;
0481         786      ;
0482         787      ;
0483         788      ;
0484         789      ;
0485         790      ;
0486         791      ;
0487         792      ;
0488         793      ;
0489         794      ;
0490         795      ;
0491         796      ;
0492         797      ;
0493         798      ;
0494         799      ;
0495         800      ;
0496         801      ;
0497         802      ;
0498         803      ;
0499         804      ;
0500         805      ;
0501         806      ;
0502         807      ;
0503         808      ;
0504         809      ;
0505         810      ;
0506         811      ;
0507         812      ;
0508         813      ;
0509         814      ;
0510         815      ;
0511         816      ;
0512         817      ;
0513         818      ;
0514         819      ;
0515         820      ;
0516         821      ;
0517         822      ;
0518         823      ;
0519         824      ;
0520         825      ;
0521         826      ;
0522         827      ;
0523         828      ;
0524         829      ;
0525         830      ;
0526         831      ;
0527         832      ;
0528         833      ;
0529         834      ;
0530         835      ;
0531         836      ;
0532         837      ;
0533         838      ;
0534         839      ;
0535         840      ;
0536         841      ;
0537         842      ;
0538         843      ;
0539         844      ;
0540         845      ;
0541         846      ;
0542         847      ;
0543         848      ;
0544         849      ;
0545         850      ;
0546         851      ;
0547         852      ;
0548         853      ;
0549         854      ;
0550         855      ;
0551         856      ;
0552         857      ;
0553         858      ;
0554         859      ;
0555         860      ;
0556         861      ;
0557         862      ;
0558         863      ;
0559         864      ;
0560         865      ;
0561         866      ;
0562         867      ;
0563         868      ;
0564         869      ;
0565         870      ;
0566         871      ;
0567         872      ;
0568         873      ;
0569         874      ;
0570         875      ;
0571         876      ;
0572         877      ;
0573         878      ;
0574         879      ;
0575         880      ;
0576         881      ;
0577         882      ;
0578         883      ;
0579         884      ;
0580         885      ;
0581         886      ;
0582         887      ;
0583         888      ;
0584         889      ;
0585         890      ;
0586         891      ;
0587         892      ;
0588         893      ;
0589         894      ;
0590         895      ;
0591         896      ;
0592         897      ;
0593         898      ;
0594         899      ;
0595         900      ;
0596         901      ;
0597         902      ;
0598         903      ;
0599         904      ;
0600         905      ;
0601         906      ;
0602         907      ;
0603         908      ;
0604         909      ;
0605         910      ;
0606         911      ;
0607         912      ;
0608         913      ;
0609         914      ;
0610         915      ;
0611         916      ;
0612         917      ;
0613         918      ;
0614         919      ;
0615         920      ;
0616         921      ;
0617         922      ;
0618         923      ;
0619         924      ;
0620         925      ;
0621         926      ;
0622         927      ;
0623         928      ;
0624         929      ;
0625         930      ;
0626         931      ;
0627         932      ;
0628         933      ;
0629         934      ;
0630         935      ;
0631         936      ;
0632         937      ;
0633         938      ;
0634         939      ;
0635         940      ;
0636         941      ;
0637         942      ;
0638         943      ;
0639         944      ;
0640         945      ;
0641         946      ;
0642         947      ;
0643         948      ;
0644         949      ;
0645         950      ;
0646         951      ;
0647         952      ;
0648         953      ;
0649         954      ;
0650         955      ;
0651         956      ;
0652         957      ;
0653         958      ;
0654         959      ;
0655         960      ;
0656         961      ;
0657         962      ;
0658         963      ;
0659         964      ;
0660         965      ;
0661         966      ;
0662         967      ;
0663         968      ;
0664         969      ;
0665         970      ;
0666         971      ;
0667         972      ;
0668         973      ;
0669         974      ;
0670         975      ;
0671         976      ;
0672         977      ;
0673         978      ;
0674         979      ;
0675         980      ;
0676         981      ;
0677         982      ;
0678         983      ;
0679         984      ;
0680         985      ;
0681         986      ;
0682         987      ;
0683         988      ;
0684         989      ;
0685         990      ;
0686         991      ;
0687         992      ;
0688         993      ;
0689         994      ;
0690         995      ;
0691         996      ;
0692         997      ;
0693         998      ;
0694         999      ;
0695         1000     ;

```

270531-17

intel

AB-41

1671

LOC	OBJ	LINE	SOURCE
0210	C3	528	CLR C
0211	742C	529	MOV A, #FULL_BIT_LOW
0213	25EC	530	ADD A, CCAP2L
0215	F5EC	531	MOV CCAP2L, A
0217	7402	532	MOV A, #FULL_BIT_HIGH
0219	35FC	533	ADDC A, CCAF2H
021B	F5FC	534	MOV CCAF2H, A
021D	D0D0	535	POP PSW
021F	D0E0	536	POP ACC
0221	32	537	RETI
0222	D55212	538	
		539	RCV_BYTE_2: DJNZ RCV_COUNT_2, RCV_DATA_2
		540	
0225	309528	541	RCV_STOP_2: JNB P1.5, ERROR_2
0228	855150	542	MOV RCV_BUF_2, RCV_REG_2
022B	D211	543	SETB RCV_DONE_2
022D	D28F	544	SETB TF1
022F	75DC11	545	MOV CCAPM2, #NEG_EDGE
0232	D0D0	546	POP PSW
0234	D0E0	547	POP ACC
0236	32	548	RETI
		549	
0237	A295	550	RCV_DATA_2: MOV C, P1.5
0239	E551	551	MOV A, RCV_REG_2
023B	13	552	RRC A
023C	F551	553	MOV RCV_REG_2, A
023E	C3	554	CLR C
023F	742C	555	MOV A, #FULL_BIT_LOW
0241	25EC	556	ADD A, CCAP2L
0243	F5EC	557	MOV CCAP2L, A
0245	7402	558	MOV A, #FULL_BIT_HIGH
0247	35FC	559	ADDC A, CCAF2H
0249	F5FC	560	MOV CCAF2H, A
024B	D0D0	561	POP PSW
024D	D0E0	562	POP ACC
024F	32	563	RETI
		564	
0250	C2B7	565	ERROR_2: CLR P3.7
0252	75DC11	566	MOV CCAPM2, #NEG_EDGE
0255	C210	567	CLR RCV_START_BIT_2
0257	D0D0	568	POP PSW
0259	D0E0	569	POP ACC
025B	32	570	RETI
		571	
		572	
		573	
		574	
		575	; This routine simulates the "RI" interrupt. When a byte is received on one
		576	; of the channels, this interrupt is generated. Bits are set so the main
		577	; routine knows which channel received a byte.
		578	
		579	
025C	C0E0	580	RECEIVE_DONE: PUSH ACC
025E	C0D0	581	PUSH PSW
0260	C28F	582	CLR TF1

270531-18

LOC	OBJ	LINE	SOURCE
0262	300106	583	JNB RCV DONE 0, RCV_1 ; Check which module received a byte
0265	C201	584	CLR RCV DONE 0 ; Clear flags needed for next reception
0267	C200	585	CLR RCV START BIT 0
0269	D202	586	SETB RCV_ON 0 ; Tell main routine which channel received
		587	; a byte
026B	300906	588	RCV_1: JNB RCV DONE 1, RCV_2
026E	C209	589	CLR RCV DONE 1
0270	C208	590	CLR RCV START BIT 1
0272	D20A	591	SETB RCV_ON 1
		592	
0274	301106	593	RCV_2: JNB RCV DONE 2, RETURN
0277	C211	594	CLR RCV DONE 2
0279	C210	595	CLR RCV START BIT 2
027B	D212	596	SETB RCV_ON 2
		597	
027D	D0D0	598	RETURN: POP PSW
027F	D0E0	599	POP ACC
0281	32	600	RETI
		601	
		602	
		603	
		604	
		605	SERIAL PORT INTERRUPT
		606	=====
		607	
		608	; When a byte is received on the full-duplex serial port, it is then
		609	; transmitted back to a "dummy" terminal. This terminal checks that the
		610	; byte it transmitted to the PCA is the same value it receives back.
		611	
		612	
0282	C0E0	613	SERIAL_PORT: PUSH ACC
0284	C0D0	614	PUSH PSW
0286	30980B	615	JNB RI, TXM ; Check whether RI or TI
0289	E599	616	MOV A, SBUF ; caused the interrupt
028B	C298	617	CLR RI
028D	F599	618	MOV SBUF, A
028F	D0D0	619	POP PSW
0291	D0E0	620	POP ACC
0293	32	621	RETI
		622	
0294	C299	623	TXM: CLR TI
0296	D0D0	624	POP PSW
0298	D0E0	625	POP ACC
029A	32	626	RETI
		627	
		628	END

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

270531-19

intel

AB-41

DOS 3.20 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
 OBJECT MODULE PLACED IN SWPORT.OBJ
 ASSEMBLER INVOKED BY: C:\AEDIT\ASM51.EXE SWPORT.TR

LOC	OBJ	LINE	SOURCE
		1	\$NOMOD51
		2	\$NOSYMBOLS
		3	\$NOLIST
		152	;
		153	;
		154	;
		155	;
		156	;
		157	;
		158	;
		159	;
		160	;
		161	;
		162	;
		163	;
0000		164	ORG 00H
0000	020036	165	LJMP INIT_TXM
		166	;
0023		167	ORG 0023H
0023	02014B	168	LJMP SERIAL_PORT
		169	;
0033		170	ORG 0033H
0033	0200D0	171	LJMP TRANSMIT
		172	;
		173	;
		174	;
		175	;
		176	;
0003		177	TXM_START_BIT_0 BIT 20H.3 ; Indicates start bit has been
000B		178	TXM_START_BIT_1 BIT 21H.3 ; transmitted
0013		179	TXM_START_BIT_2 BIT 22H.3 ;
		180	;
0004		181	TXM_IN_PROGRESS_0 BIT 20H.4 ; Indicates transmit is in progress
000C		182	TXM_IN_PROGRESS_1 BIT 21H.4 ;
0014		183	TXM_IN_PROGRESS_2 BIT 22H.4 ;
		184	;
0034		185	TXM_BUF_0 DATA 34H ; Software transmit "SBUF"
0044		186	TXM_BUF_1 DATA 44H ;
0054		187	TXM_BUF_2 DATA 54H ;
		188	;
0035		189	TXM_REG_0 DATA 35H ; Temporary register for
0045		190	TXM_REG_1 DATA 45H ; transmitting bits
0055		191	TXM_REG_2 DATA 55H ;
		192	;
0036		193	TXM_COUNT_0 DATA 36H ; Counter for transmitting bits
0046		194	TXM_COUNT_1 DATA 46H ;
0056		195	TXM_COUNT_2 DATA 56H ;
		196	;
0037		197	DATA_0 DATA 37H ; Register used for the test
0047		198	DATA_1 DATA 47H ; program

01/01/90 BYCS 270531-21

01/01/80 PAGE 3

```

LOC  OBJ      LINE      SOURCE
0084  02009D    254      ;
0085  02009D    255      ;
0086  02009D    256      ;
0087  02009D    257      ;
0088  02009D    258      ;
0089  02009D    259      ;
0090  02009D    260      ;
0091  02009D    261      ;
0092  02009D    262      ;
0093  02009D    263      ;
0094  02009D    264      ;
0095  02009D    265      ;
0096  02009D    266      ;
0097  02009D    267      ;
0098  02009D    268      ;
0099  02009D    269      ;
0100  02009D    270      ;
0101  02009D    271      ;
0102  02009D    272      ;
0103  02009D    273      ;
0104  02009D    274      ;
0105  02009D    275      ;
0106  02009D    276      ;
0107  02009D    277      ;
0108  02009D    278      ;
0109  02009D    279      ;
0110  02009D    280      ;
0111  02009D    281      ;
0112  02009D    282      ;
0113  02009D    283      ;
0114  02009D    284      ;
0115  02009D    285      ;
0116  02009D    286      ;
0117  02009D    287      ;
0118  02009D    288      ;
0119  02009D    289      ;
0120  02009D    290      ;
0121  02009D    291      ;
0122  02009D    292      ;
0123  02009D    293      ;
0124  02009D    294      ;
0125  02009D    295      ;
0126  02009D    296      ;
0127  02009D    297      ;
0128  02009D    298      ;
0129  02009D    299      ;
0130  02009D    300      ;
0131  02009D    301      ;
0132  02009D    302      ;
0133  02009D    303      ;
0134  02009D    304      ;
0135  02009D    305      ;
0136  02009D    306      ;
0137  02009D    307      ;
0138  02009D    308      ;
0139  02009D    309      ;
0140  02009D    310      ;
0141  02009D    311      ;
0142  02009D    312      ;
0143  02009D    313      ;
0144  02009D    314      ;
0145  02009D    315      ;
0146  02009D    316      ;
0147  02009D    317      ;
0148  02009D    318      ;
0149  02009D    319      ;
0150  02009D    320      ;
0151  02009D    321      ;
0152  02009D    322      ;
0153  02009D    323      ;
0154  02009D    324      ;
0155  02009D    325      ;
0156  02009D    326      ;
0157  02009D    327      ;
0158  02009D    328      ;
0159  02009D    329      ;
0160  02009D    330      ;
0161  02009D    331      ;
0162  02009D    332      ;
0163  02009D    333      ;
0164  02009D    334      ;
0165  02009D    335      ;
0166  02009D    336      ;
0167  02009D    337      ;
0168  02009D    338      ;
0169  02009D    339      ;
0170  02009D    340      ;
0171  02009D    341      ;
0172  02009D    342      ;
0173  02009D    343      ;
0174  02009D    344      ;
0175  02009D    345      ;
0176  02009D    346      ;
0177  02009D    347      ;
0178  02009D    348      ;
0179  02009D    349      ;
0180  02009D    350      ;
0181  02009D    351      ;
0182  02009D    352      ;
0183  02009D    353      ;
0184  02009D    354      ;
0185  02009D    355      ;
0186  02009D    356      ;
0187  02009D    357      ;
0188  02009D    358      ;
0189  02009D    359      ;
0190  02009D    360      ;
0191  02009D    361      ;
0192  02009D    362      ;
0193  02009D    363      ;
0194  02009D    364      ;
0195  02009D    365      ;
0196  02009D    366      ;
0197  02009D    367      ;
0198  02009D    368      ;
0199  02009D    369      ;
0200  02009D    370      ;
0201  02009D    371      ;
0202  02009D    372      ;
0203  02009D    373      ;
0204  02009D    374      ;
0205  02009D    375      ;
0206  02009D    376      ;
0207  02009D    377      ;
0208  02009D    378      ;
0209  02009D    379      ;
0210  02009D    380      ;
0211  02009D    381      ;
0212  02009D    382      ;
0213  02009D    383      ;
0214  02009D    384      ;
0215  02009D    385      ;
0216  02009D    386      ;
0217  02009D    387      ;
0218  02009D    388      ;
0219  02009D    389      ;
0220  02009D    390      ;
0221  02009D    391      ;
0222  02009D    392      ;
0223  02009D    393      ;
0224  02009D    394      ;
0225  02009D    395      ;
0226  02009D    396      ;
0227  02009D    397      ;
0228  02009D    398      ;
0229  02009D    399      ;
0230  02009D    400      ;
0231  02009D    401      ;
0232  02009D    402      ;
0233  02009D    403      ;
0234  02009D    404      ;
0235  02009D    405      ;
0236  02009D    406      ;
0237  02009D    407      ;
0238  02009D    408      ;
0239  02009D    409      ;
0240  02009D    410      ;
0241  02009D    411      ;
0242  02009D    412      ;
0243  02009D    413      ;
0244  02009D    414      ;
0245  02009D    415      ;
0246  02009D    416      ;
0247  02009D    417      ;
0248  02009D    418      ;
0249  02009D    419      ;
0250  02009D    420      ;
0251  02009D    421      ;
0252  02009D    422      ;
0253  02009D    423      ;
0254  02009D    424      ;
0255  02009D    425      ;
0256  02009D    426      ;
0257  02009D    427      ;
0258  02009D    428      ;
0259  02009D    429      ;
0260  02009D    430      ;
0261  02009D    431      ;
0262  02009D    432      ;
0263  02009D    433      ;
0264  02009D    434      ;
0265  02009D    435      ;
0266  02009D    436      ;
0267  02009D    437      ;
0268  02009D    438      ;
0269  02009D    439      ;
0270  02009D    440      ;
0271  02009D    441      ;
0272  02009D    442      ;
0273  02009D    443      ;
0274  02009D    444      ;
0275  02009D    445      ;
0276  02009D    446      ;
0277  02009D    447      ;
0278  02009D    448      ;
0279  02009D    449      ;
0280  02009D    450      ;
0281  02009D    451      ;
0282  02009D    452      ;
0283  02009D    453      ;
0284  02009D    454      ;
0285  02009D    455      ;
0286  02009D    456      ;
0287  02009D    457      ;
0288 
```

270531-22

MCS-51 MACRO ASSEMBLER SWPORT

01/01/80 PAGE 4

LOC	OBJ	LINE	SOURCE
00E3	D53607	309	TXM_BYTE_0: DJNZ TXM_COUNT_0, TXM_DATA_0 ; If bit count equals 1 thru 9,
00E6	D2B2	310	TXM_STOP_0: ; Transmit data bits (8 total)
00E8	C204	311	SETB P3.2 ; When bit count = 0, transmit stop bit
00EA	0200F7	312	CLR TXM_IN_PROGRESS_0 ; Indicate transmission is finished and
00ED	E535	313	JMP TRANSMIT_1 ; ready for next byte
00EF	13	314	TXM_DATA_0: MOV A, TXM_REG_0 ; Transmit one bit at a time
00F0	92B2	315	RRC A ; through the carry bit
00F2	F535	316	MOV P3.2, C ; Save what's not been sent
00F4	0200F7	317	MOV TXM_REG_0, A ; Check next transmit pin
		318	JMP TRANSMIT_1
		319	;
		320	;
		321	;
		322	CHANNEL 1
		323	-----
00F7	300C1E	324	TRANSMIT_1: JNB TXM_IN_PROGRESS_1, TRANSMIT_2 ; Similar to TRANSMIT_0
00FA	20B07	325	JB TXM_START_BIT_1, TXM_BYTE_1
00FD	C2B3	326	CLR P3.3
00FF	D20B	327	SETB TXM_START_BIT_1
0101	020118	328	JMP TRANSMIT_2
0104	D54607	329	TXM_BYTE_1: DJNZ TXM_COUNT_1, TXM_DATA_1
0107	D2B3	330	TXM_STOP_1: SETB P3.3
0109	C20C	331	CLR TXM_IN_PROGRESS_1
010B	020118	332	JMP TRANSMIT_2
010E	E545	333	TXM_DATA_1: MOV A, TXM_REG_1
0110	13	334	RRC A
0111	92B3	335	MOV P3.3, C
0113	F545	336	MOV TXM_REG_1, A
0115	020118	337	JMP TRANSMIT_2
		338	;
		339	;
		340	CHANNEL 2
		341	-----
0118	30141E	342	TRANSMIT_2: JNB TXM_IN_PROGRESS_2, TXM_EXIT ; Similar to TRANSMIT_0
011B	201307	343	JB TXM_START_BIT_2, TXM_BYTE_2
011E	C2B4	344	CLR P3.4
0120	D213	345	SETB TXM_START_BIT_2
0122	020139	346	JMP TXM_EXIT
0125	D55607	347	TXM_BYTE_2: DJNZ TXM_COUNT_2, TXM_DATA_2
0128	D2B4	348	TXM_STOP_2: SETB P3.4
012A	C214	349	CLR TXM_IN_PROGRESS_2
012C	020139	350	JMP TXM_EXIT
012F	E555	351	TXM_DATA_2: MOV A, TXM_REG_2
0131	13	352	RRC A
0132	92B4	353	MOV P3.4, C
0134	F555	354	MOV TXM_REG_2, A
0136	020139	355	JMP TXM_EXIT
		356	;
		357	;
		358	;
		359	;
		360	;
		361	;
		362	;
		363	;

270531-23

MCS-51 MACRO ASSEMBLER

SWPORT

01/01/80 PAGE 5

LOC	OBJ	LINE	SOURCE
0139	C3	364	TXM_EXIT: CLR C
013A	742C	365	MOV A, #FULL_BIT_LOW ; Update compare value with
013C	25ED	366	ADD A, CCAP3L ; full bit time = 22CH
013E	F5ED	367	MOV CCAP3L, A
0140	7402	368	MOV A, #FULL_BIT_HIGH
0142	35FD	369	ADDC A, CCAP3H
0144	F5FD	370	MOV CCAP3H, A
0146	D0D0	371	POP PSW
0148	D0E0	372	POP ACC
014A	32	373	RETI
		374	
		375	
		376	SERIAL PORT INTERRUPT
		377	=====
		378	
		379	; When a byte is received on the full-duplex serial port, it is then
		380	; transmitted back to a "dummy" terminal. This terminal checks that
		381	; the byte it transmitted to the PCA is the same value it receives back.
		382	
		383	
014B	C0E0	384	SERIAL_PORT: PUSH ACC
014D	C0D0	385	PUSH PSW
014F	30980B	386	JNB RI, TXM ; Check whether RI or TI
0152	E599	387	MOV A, SBUF ; caused the interrupt
0154	C298	388	CLR RI
0156	F599	389	MOV SBUF, A
0158	D0D0	390	POP PSW
015A	D0E0	391	POP ACC
015C	32	392	RETI
		393	
015D	C299	394	TXM: CLR TI
015F	D0D0	395	POP PSW
0161	D0E0	396	POP ACC
0163	32	397	RETI
		398	
		399	
		400	END

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

270531-24



APPLICATION NOTE

AP-415

July 1988

2

83C51FA/FB PCA Cookbook

BETSY JONES
ECO APPLICATIONS ENGINEER

Order Number: 270609-001

83C51FA/FB PCA COOKBOOK

CONTENTS

PAGE

PCA OVERVIEW 2-248

PCA TIMER/COUNTER 2-248

COMPARE/CAPTURE MODULES 2-250

CAPTURE MODE 2-252

Measuring Pulse Widths 2-252

Measuring Periods 2-254

Measuring Frequencies 2-254

Measuring Duty Cycles 2-256

Measuring Phase Differences 2-257

Reading the PCA Timer 2-260

COMPARE MODE 2-260

SOFTWARE TIMER 2-260

HIGH SPEED OUTPUT 2-262

WATCHDOG TIMER 2-265

PULSE WIDTH MODULATOR 2-266

CONCLUSION 2-268

APPENDICES

A. Test Routines 2-269

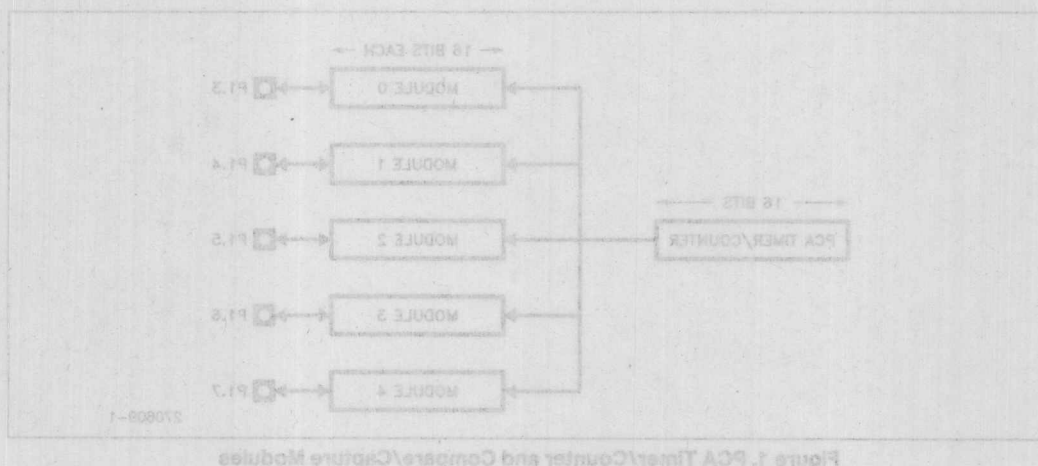
B. Duty Cycle Calculation 2-286

C. Special Function Registers 2-289

FIGURES	PAGE
1. PCA Timer/Counter and Compare/Capture Modules	2-248
2. PCA Interrupt	2-251
3. PCA Capture Mode	2-252
4. Measuring Pulse Width	2-252
5. Measuring Period	2-254
6. Measuring Frequency	2-254
7. Measuring Duty Cycle	2-256
8. Measuring Phase Differences	2-257
9. Software Timer Mode	2-260
10. High Speed Output Mode	2-262
11. Watchdog Timer Mode	2-265
12. PWM Mode	2-266
13. CCAPnH Varies Duty Cycle	2-267

LISTINGS	PAGE
1. Measuring Pulse Widths	2-253
2. Measuring Frequencies	2-255
3. Measuring Duty Cycle	2-256
4. Measuring Phase Differences	2-258
5. Software Timer	2-261
6. High Speed Output (Without Interrupt)	2-262
7. High Speed Output (With Interrupt)	2-263
8. High Speed Output (Single Pulse)	2-264
9. Watchdog Timer	2-266
10. PWM	2-268

TABLES	PAGE
1. PCA Timer/Counter Inputs	2-249
2. CMOD Values	2-249
3. Compare/Capture Mode Values	2-250
4. PWM Frequencies	2-267



The Programmable Counter Array (PCA) which are available on the 83C51FA and 83C51FB. Included are cookbook samples of code in typical applications to simplify the use of the PCA. Since all the examples are written in assembly language, it is assumed the reader is familiar with ASM51. For further information on these products or ASM51 refer to the Embedded Controller Handbook (Vol. I).

PCA OVERVIEW

The major new feature on the 83C51FA and 83C51FB is the Programmable Counter Array. The PCA provides more timing capabilities with less CPU intervention than the standard timer/counters. Its advantages include reduced software overhead and improved accuracy.

The PCA consists of a dedicated timer/counter which serves as the time base for an array of five compare/capture modules. Figure 1 shows a block diagram of the PCA. Notice that the PCA timer and modules are all 16-bits. If an external event is associated with a module, that function is shared with the corresponding Port 1 pin. If the module is not using the port pin, the pin can still be used for standard I/O.

one of the following modes:

- Rising and/or Falling Edge Capture
- Software Timer
- High Speed Output
- Watchdog Timer (Module 4 only)
- Pulse Width Modulator.

All of these modes will be discussed later in detail. However, let's first look at how to set up the PCA timer and modules.

PCA TIMER/COUNTER

The timer/counter for the PCA is a free-running 16-bit timer consisting of registers CH and CL (the high and low bytes of the count values). It is the only timer which can service the PCA. The clock input can be selected from the following four modes:

- oscillator frequency \div 12 (Mode 0)
- oscillator frequency \div 4 (Mode 1)
- Timer 0 overflows (Mode 2)
- external input on P1.2 (Mode 3)

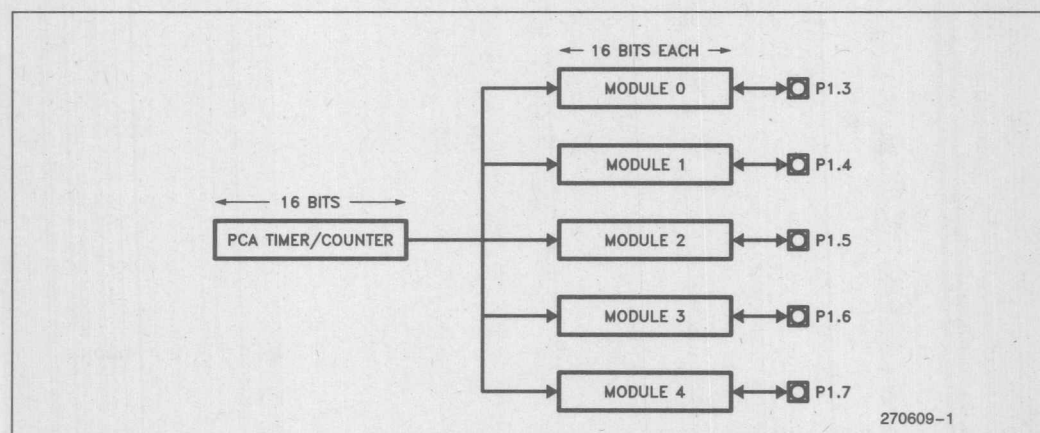


Figure 1. PCA Timer/Counter and Compare/Capture Modules

The table below summarizes the various clock inputs for each mode at two common frequencies. In Mode 0, the clock input is simply a machine cycle count, whereas in Mode 1 the input is clocked three times faster. In Mode 2, Timer 0 overflows are counted allowing for a range of slower inputs to the timer. And finally, if the input is external the PCA timer counts 1-to-0 transitions with the maximum clock frequency equal to $\frac{1}{8}$ x oscillator frequency.

Table 1. PCA Timer/Counter Inputs

PCA Timer/Counter Mode	Clock Increments	
	12 MHz	16 MHz
Mode 0: fosc / 12	1 μ sec	0.75 μ sec
Mode 1: fosc / 4	330 nsec	250 nsec
Mode 2*: Timer 0 Overflows Timer 0 programmed in:		
8-bit mode	256 μ sec	192 μ sec
16-bit mode	65 msec	49 msec
8-bit auto-reload	1 to 255 μ sec	0.75 to 191 μ sec
Mode 3: External Input MAX	0.66 μ sec	0.50 μ sec

*In Mode 2, the overflow interrupt for Timer 0 does not need to be enabled.

Special Function Register CMOD contains the Count Pulse Select bits (CPS1 and CPS0) to specify the PCA timer input. This register also contains the ECF bit which enables an interrupt when the counter overflows. In addition, the user has the option of turning off the PCA timer during Idle Mode by setting the Counter Idle bit (CIDL). This can further reduce power consumption by an additional 30%.

CMOD: Counter Mode Register

CIDL	WDTE	—	—	—	CPS1	CPS0	ECF
------	------	---	---	---	------	------	-----

Address = 0D9H

Not Bit Addressable

Reset Value = 00XX X000B

NOTE:

The user should write 0s to unimplemented bits. These bits may be used in future MCS-51 products to invoke new features, and in that case the inactive value of the new bit will be 0. When read, these bits must be treated as don't-cares.

Table 2 lists the values for CMOD in the four possible timer modes with and without the overflow interrupt enabled. This list assumes that the PCA will be left running during Idle Mode.

Table 2. CMOD Values

PCA Count Pulse Selected	CMOD value	
	without interrupt enabled	with interrupt enabled
Internal clock, Fosc/12	00 H	01 H
Internal clock, Fosc/ 4	02 H	03 H
Timer 0 overflow	04H	05 H
External clock at P1.2	06 H	07 H

The CCON register shown below contains the Counter Run bit (CR) which turns the timer on or off. When the PCA timer overflows, the Counter Overflow bit (CF) gets set. CCON also contains the five event flags for the PCA modules. The purpose of these flags will be discussed in the next section.

CCON: Counter Control Register

CF	CR	—	CCF4	CCF3	CCF2	CCF1	CCF0
Address = 0D8H					Reset Value = 00X0 0000B		
Bit Addressable							

The PCA timer registers (CH and CL) can be read and written to at any time. However, to read the full 16-bit timer value simultaneously requires using one of the PCA modules in the capture mode and toggling a port pin in software. More information on reading the PCA timer is provided in the section on the Capture Mode.

COMPARE/CAPTURE MODULES

Each of the five compare/capture modules has a mode register called CCAPMn (n = 0,1,2,3,or 4) to select which function it will perform. Note the ECCFn bit which enables an interrupt to occur when a module's event flag is set.

CCAPMn: Compare/Capture Mode Register

—	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn
Address = 0DAH (n=0)				Reset Value = X000 0000B			
0DBH (n=1)							
0DCH (n=2)							
0DDH (n=3)							
0DEH (n=4)							

Table 3 lists the CCAPMn values for each different mode with and without the PCA interrupt enabled; that is, the interrupt is optional for all modes. However, some of the PCA modes require software servicing. For example, the Capture modes need an interrupt so that back-to-back events can be recognized. Also, in most applications the purpose of the Software Timer mode is to generate interrupts in software so it would be useless not to have the interrupt enabled. The PWM mode, on the other hand, does not require CPU intervention so the interrupt is normally not enabled.

Table 3. Compare/Capture Mode Values

Module Function	CCAPMn Value	
	without interrupt enabled	with interrupt enabled
Capture Positive only	20H	21 H
Capture Negative only	10H	11 H
Capture Pos. or Neg.	30H	31 H
Software Timer	48H	49 H
High Speed Output	4C H	4D H
Watchdog Timer	48 or 4C H	—
Pulse Width Modulator	42 H	43H

It should be mentioned that a particular module can change modes within the program. For example, a module might be used to sample incoming data. Initially it could be set up to capture a falling edge transition. Then the same module can be reconfigured as a software timer to interrupt the CPU at regular intervals and sample the pin.

Each module also has a pair of 8-bit compare/capture registers (CCAPnH, CCAPnL) associated with it. These registers are used to store the time when a capture event occurred or when a compare event should occur. Remember, event times are based on the free-running PCA timer (CH and CL). For the PWM mode, the high byte register CCAPnH controls the duty cycle of the waveform.

When an event occurs, a flag in CCON is set for the appropriate module. This register is bit addressable so that event flags can be checked individually.

CCON: Counter Control Register

CF	CR	—	CCF4	CCF3	CCF2	CCF1	CCF0
----	----	---	------	------	------	------	------

Address = 0D8H

Reset Value = 00X0 0000B

Bit Addressable

These five event flags plus the PCA timer overflow flag share an interrupt vector as shown below. These flags are not cleared when the hardware vectors to the PCA interrupt address (0033H) so that the user can determine which event caused the interrupt. This also allows the user to define the priority of servicing each module.

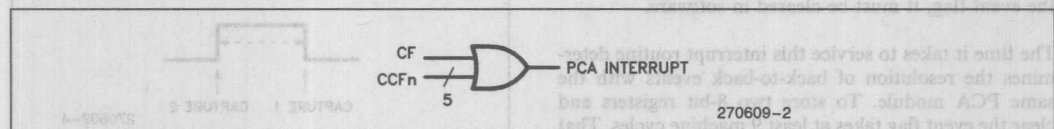


Figure 2. PCA Interrupt

An additional bit was added to the Interrupt Enable (IE) register for the PCA interrupt. Similarly, a high priority bit was added to the Interrupt Priority (IP) register.

IE: Interrupt Enable Register

EA	EC	ET2	ES	ET1	EX1	ET0	EX0
----	----	-----	----	-----	-----	-----	-----

Address = 0A8H

Reset Value = 0000 0000B

Bit Addressable

IP: Interrupt Priority Register

—	PPC	PT2	PS	PT1	PX1	PT0	PX0
---	-----	-----	----	-----	-----	-----	-----

Address = 0B8H

Reset Value = X000 0000B

Bit Addressable

Remember, each of the six possible sources for the PCA interrupt must be individually enabled as well—in the CCAPMn register for the modules and in the CCON register for the timer.

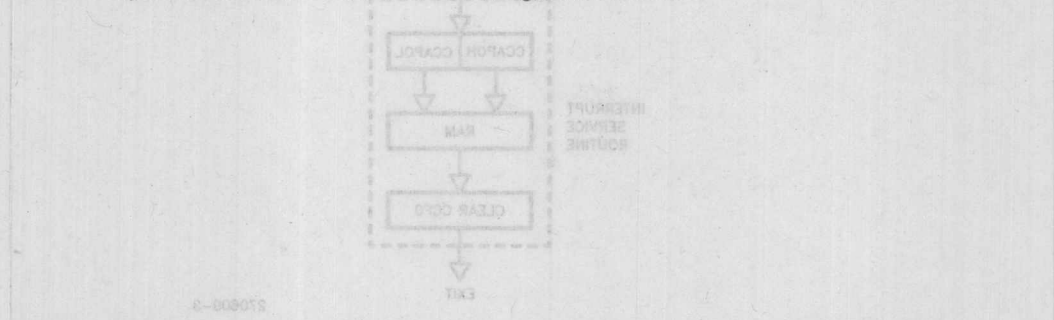


Figure 3. PCA Capture Mode (Module 0)

CAPTURE MODE

Both positive and negative transitions can trigger a capture with the PCA. This allows the PCA flexibility to measure periods, pulse widths, duty cycles, and phase differences on up to five separate inputs. This section gives examples of all these different applications.

Figure 3 shows how the PCA handles a capture event. Using Module 0 for this example, the signal is input to P1.3. When a transition is detected on that pin, the 16-bit value of the PCA timer (CH,CL) is loaded into the capture registers (CCAP0H,CCAP0L). Module 0's event flag is set and an interrupt is flagged. The interrupt will then be generated if it has been properly enabled.

In the interrupt service routine, the 16-bit capture value must be saved in RAM before the next event capture occurs; a subsequent capture will write over the first capture value. Also, since the hardware does not clear the event flag, it must be cleared in software.

The time it takes to service this interrupt routine determines the resolution of back-to-back events with the same PCA module. To store two 8-bit registers and clear the event flag takes at least 9 machine cycles. That includes the call to the interrupt routine. At 12-MHz, this routine would take less than 10 microseconds. However, depending on the frequency and interrupt latency, the resolution will vary with each application.

Measuring Pulse Widths

To measure the pulse width of a signal, the PCA module must capture both rising and falling edges (see Figure 4). The module can be programmed to capture either edge if it is known which edge will occur first. However, if this is not known, the user can select which edge will trigger the first capture by choosing the proper mode for the module.

Listing 1 shows an example of measuring pulse widths. (It's assumed the incoming signal matches the one in Figure 4.) In the interrupt routine the first set of capture values are stored in RAM. After the second capture, a subtraction routine calculates the pulse width in units of PCA timer ticks. Note that the subtraction does not have to be completed in the interrupt service routine. Also, this example assumes that the two capture events will occur within 2^{16} counts of the PCA timer, i.e. rollovers of the PCA timer are not counted.

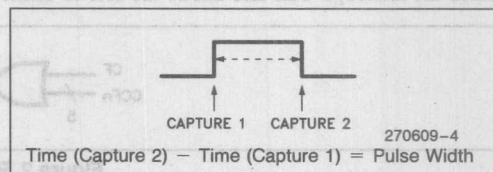


Figure 4. Measuring Pulse Width

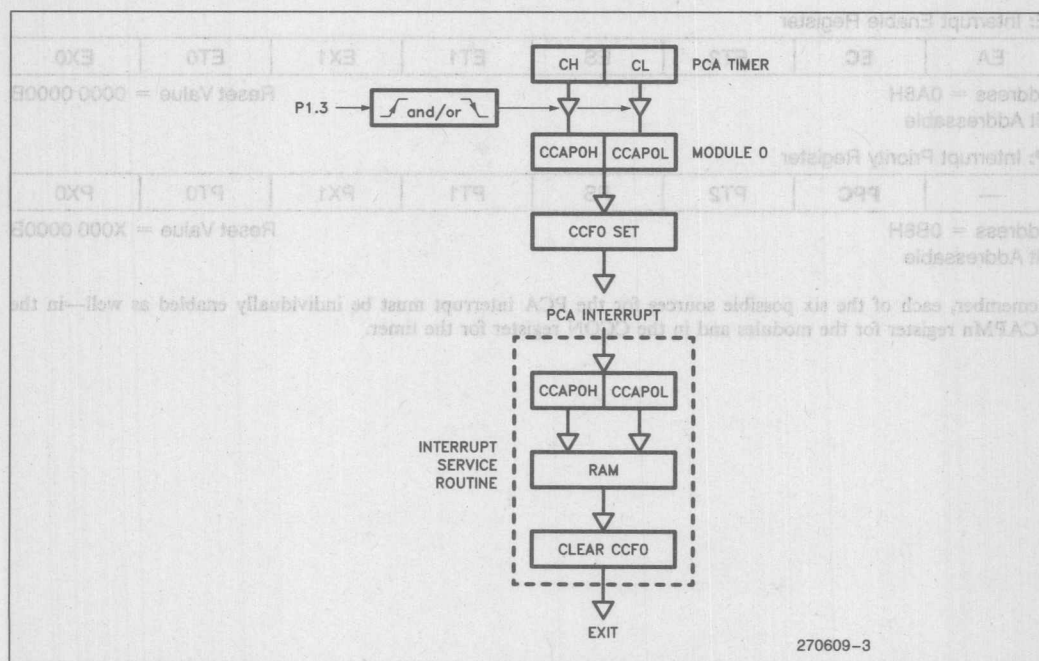


Figure 3. PCA Capture Mode (Module 0)

Listing 1. Measuring Pulse Widths

```

; RAM locations to store capture values
CAPTURE      DATA    30H
PULSE_WIDTH  DATA    32H
FLAG         BIT      20H.0

ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT

PCA_INIT:
    MOV CMOD, #00H      ; Initialize PCA timer
    MOV CH, #00H        ; Input to timer = 1/12 X Fosc
    MOV CL, #00H

;
; Initialize Module 0 in capture mode
MOV CCAPMO, #21H      ; Capture positive edge first
; for measuring pulse width

;
SETB EC              ; Enable PCA interrupt
SETB EA
SETB CR              ; Turn PCA timer on
CLR FLAG             ; clear test flag

;
; *****
; Main program goes here
; *****

; This example assumes Module 0 is the only PCA module
; being used. If other modules are used, software must
; check which module's event caused the interrupt.
;
PCA_INTERRUPT:
    CLR CCFO           ; Clear Module 0's event flag
    JB FLAG, SECOND_CAPTURE ; Check if this is the first
; capture or second

FIRST_CAPTURE:
    MOV CAPTURE, CCAPOL ; Save 16-bit capture value
    MOV CAPTURE+1, CCAPOH ; in RAM
    MOV CCAPMO, #11H    ; Change module to now capture
; falling edges
    SETB FLAG           ; Signify 1st capture complete
    RETI

;
SECOND_CAPTURE:
    PUSH ACC
    PUSH PSW
    CLR C
    MOV A, CCAPOL       ; 16-bit subtract
    SUBB A, CAPTURE
    MOV PULSE_WIDTH, A ; 16-bit result stored in
    MOV A, CCAPOH       ; two 8-bit RAM locations
    SUBB A, CAPTURE+1
    MOV PULSE_WIDTH+1, A
    ;
    MOV CCAPMO, #21H    ; Optional--needed if user wants to
    CLR FLAG            ; measure next pulse width
    POP PSW
    POP ACC
    RETI

```

Measuring Periods

Measuring the period of a signal with the PCA is similar to measuring the pulse width. The only difference will be the trigger source for the capture mode. In Figure 5, rising edges are captured to calculate the period. The code is identical to Listing 1 except that the capture mode should not be changed in the interrupt routine. The result of the subtraction will be the period.

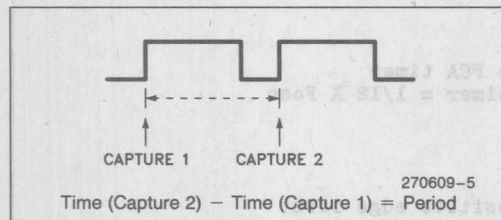


Figure 5. Measuring Period

Measuring Frequencies

Measuring a frequency with the PCA capture mode involves calculating a sample time for a known number of samples. In Figure 6, the time between the first capture and the "Nth" capture equals the sample time T. Listing 2 shows the code for N = 10 samples. It's assumed that the sample time is less than 2¹⁶ counts of the PCA timer.

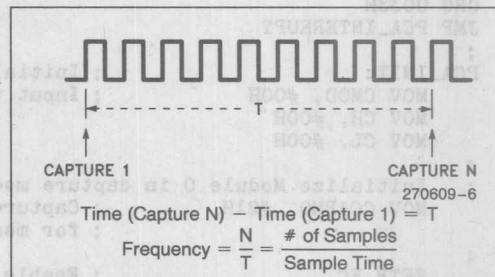


Figure 6. Measuring Frequency


```

; RAM locations to store capture values
CAPTURE DATA 30H
PERIOD DATA 32H
SAMPLE_COUNT DATA 34H
FLAG BIT 20H.0

;
ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT
;
PCA_INIT:
; Initialization of PCA timer, Module 0, and interrupt is the
; same as in Listing 1. Also need to initialize the sample
; count.
;
MOV SAMPLE_COUNT, #10D ; N = 10 for this example
;
*****
; Main program goes here
*****
;
; This code assumes only Module 0 is being used.
PCA_INTERRUPT:
CLR CCFO ; Clear module 0's event flag
JB FLAG, NEXT_CAPTURE
;
FIRST_CAPTURE:
MOV CAPTURE, CCAPOL
MOV CAPTURE+1, CCAPOH
SETB FLAG ; Signify first capture complete
RETI
;
NEXT_CAPTURE:
DJNZ SAMPLE_COUNT, EXIT
PUSH ACC
PUSH PSW
CLR C
MOV A, CCAPOL ; 16-bit subtraction
SUBB A, CAPTURE
MOV PERIOD, A
MOV A, CCAPOH
SUBB A, CAPTURE+1
MOV PERIOD+1, A
;
MOV SAMPLE_COUNT, #10D ; Reload for next period
CLR FLAG
POP PSW
POP ACC
EXIT:
RETI

```

The user may instead want to measure frequency by counting pulses for a known sample time. In this case, one module is programmed in the capture mode to count edges (either rising or falling), and a second module is programmed as a software timer to mark the sample time. An example of a software timer is given later. For information on resolution in measuring frequencies, refer to Article Reprint AR-517, "Using the 8051 Microcontroller with Resonant Transducers," in the Embedded Controller Handbook.

Measuring Duty Cycles

To measure the duty cycle of an incoming signal, both rising and falling edges need to be captured. Then the duty cycle must be calculated based on three capture values as seen in Figure 7. The same initialization routine is used from the previous example. Only the PCA interrupt service routine is given in Listing 3.

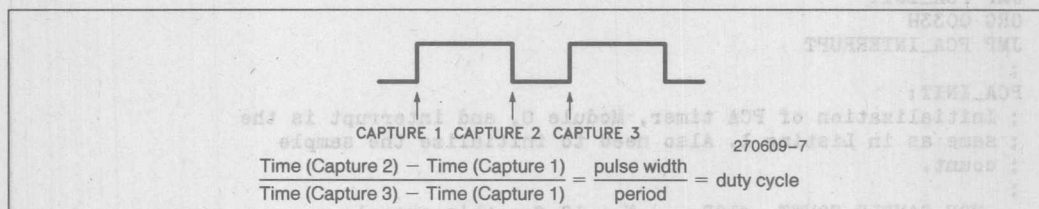


Figure 7. Measuring Duty Cycle

Listing 3. Measuring Duty Cycle

```
; RAM locations to store capture values
CAPTURE      DATA    30H
PULSE_WIDTH  DATA    32H
PERIOD       DATA    34H
FLAG_1       BIT      20H.0
FLAG_2       BIT      20H.1

;
ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT
;
PCA_INIT:
; Initialization for PCA timer, module, and interrupt the same
; as in Listing 1. Capture positive edge first, then either
; edge.
;
; *****
; Main program goes here
; *****
;
; This code assumes only Module 0 is being used.
PCA_INTERRUPT:
    CLR CCFO          ; Clear Module 0's event flag
    JB FLAG_1, SECOND_CAPTURE
;
FIRST_CAPTURE:
    MOV CAPTURE, CCAPOL
    MOV CAPTURE+1, CCAPOH
    SETB FLAG_1        ; Signify first capture complete
    MOV CCAPMO, #31H    ; Capture either edge now
    RETI
```

Listing 3. Measuring Duty Cycle (Continued)

```

;SECOND_CAPTURE:
    PUSH ACC
    PUSH PSW
    JB FLAG_2, THIRD_CAPTURE
    CLR C
    MOV A, CCAPOH
    SUBB A, CAPTURE
    MOV PULSE_WIDTH, A
    MOV A, CCAPOH
    SUBB A, CAPTURE+1
    MOV PULSE_WIDTH+1, A
    SETB FLAG_2
    POP PSW
    POP ACC
    RETI

;THIRD_CAPTURE:
    CLR C
    MOV A, CCAPOH
    SUBB A, CAPTURE
    MOV PERIOD, A
    MOV A, CCAPOH
    SUBB A, CAPTURE+1
    MOV PERIOD+1, A
    MOV CCAPMO, #21H
    CLR FLAG_1
    CLR FLAG_2
    POP PSW
    POP ACC
    RETI

```

After the third capture, a 16-bit by 16-bit divide routine needs to be executed. This routine is located in Appendix B. Due to its length, it's up to the user whether the divide routine should be completed in the interrupt routine or be called as a subroutine from the main program.

between two or more signals. For this example, two signals are input to Modules 0 and 1 as seen in Figure 8. Both modules are programmed to capture rising edges only. Listing 4 shows the code needed to measure the difference between these two signals. This code does not assume one signal is leading or lagging the other.

Measuring Phase Differences

Because the PCA modules share the same time base, the PCA is useful for measuring the phase difference

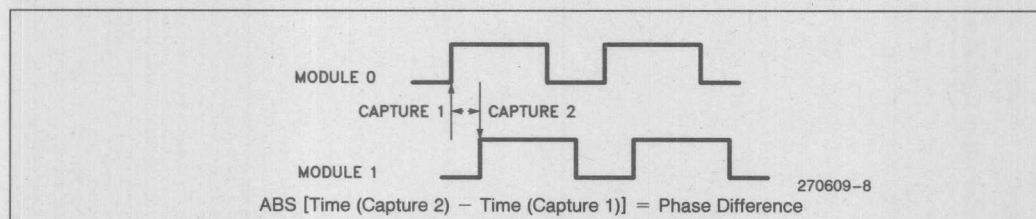


Figure 8. Measuring Phase Differences

```
; RAM locations to store capture values
```

```
CAPTURE_0      DATA      30H
CAPTURE_1      DATA      32H
PHASE          DATA      34H
FLAG_0         BIT        20H.0
FLAG_1         BIT        20H.1
```

```
;
ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT
;
```

```
PCA_INIT:
```

```
; Same initialization for PCA timer, and interrupt as
; in Listing 1. Initialize two PCA modules as follows:
```

```
MOV CCAPMO, #21H      ; Module 0 capture rising edges
MOV CCAPM1, #21H      ; Module 1 same
```

```
;
;*****
; Main program goes here
;*****
```

```
; This code assumes only Modules 0 and 1 are being used.
```

```
PCA_INTERRUPT:
```

```
JB CCFO, MODULE_0      ; Determine which module's
JB CCFL, MODULE_1      ; event caused the interrupt
```

```
;
MODULE_0:
```

```
CLR CCFO              ; Clear Module 0's event flag
MOV CAPTURE_0, CCAPOL  ; Save 16-bit capture value
MOV CAPTURE_0+1, CCAPOH
JB FLAG_1, CALCULATE_PHASE ; If capture complete on
```

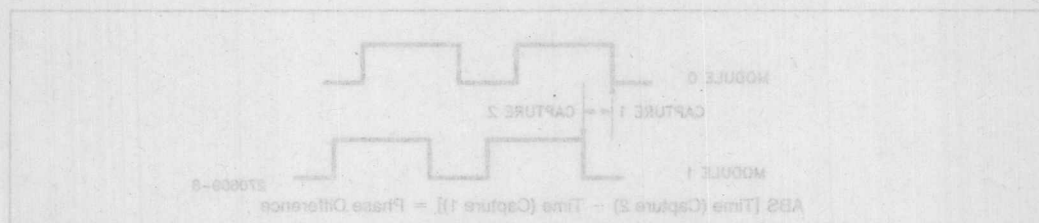
```
; Module 1, go to calculation
```

```
SETB FLAG_0          ; Signify capture on Module 0
```

```
RETI
```

Measuring Phase Differences

Because the PCA modules share the same time base, the PCA is useful for measuring the phase difference



Listing 4. Measuring Phase Differences (Continued)

```

MODULE_1:
CLR CCF_1 ; Clear Module 1's event flag
MOV CAPTURE_1, CCAP1L
MOV CAPTURE_1+1, CCAP1H
JB FLAG_0, CALCULATE_PHASE ; If capture complete on
; Module 0, go to calculation
SETB FLAG_1 ; Signify capture on Module 1
RETI

;
CALCULATE_PHASE:
PUSH ACC ; This calculation does not
PUSH PSW ; have to be completed in the
CLR C ; interrupt service routine
;
JB FLAG_0, MOD0_LEADING
JB FLAG_1, MOD1_LEADING
;
MOD0_LEADING:
MOV A, CAPTURE_1
SUBB A, CAPTURE_0
MOV PHASE, A
MOV A, CAPTURE_1+1
SUBB A, CAPTURE_0+1
MOV PHASE+1, A
CLR FLAG_0
JMP EXIT

;
MOD1_LEADING:
MOV A, CAPTURE_0
SUBB A, CAPTURE_1
MOV PHASE, A
MOV A, CAPTURE_0+1
SUBB A, CAPTURE_1+1
MOV PHASE+1, A
CLR FLAG_1

EXIT:
POP PSW
POP ACC
RETI

```

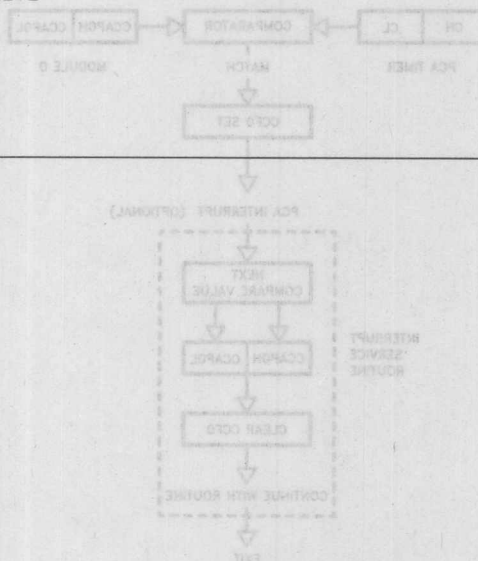


Figure 2. Software Timer Mode (Module 0)

Reading the PCA Timer

Some applications may require that the PCA timer be read instantaneously as a real-time event. Since the timer consists of two 8-bit registers (CH,CL), it would normally take two MOV instructions to read the whole timer. An invalid read could occur if the registers rolled over in the middle of the two MOVs.

However, with the capture mode a 16-bit timer value can be loaded into the capture registers by toggling a port pin. For example, configure Module 0 to capture falling edges and initialize P1.3 to be high. Then when the user wants to read the PCA timer, clear P1.3 and the full 16-bit timer value will be saved in the capture registers. It's still optional whether the user wants to generate an interrupt with the capture.

COMPARE MODE

In this mode, the 16-bit value of the PCA timer is compared with a 16-bit value pre-loaded in the module's compare registers. The comparison occurs three times per machine cycle in order to recognize the fastest possible clock input, i.e. $\frac{1}{4} \times$ oscillator frequency. When there is a match, one of three events can happen:

- (1) an interrupt — Software Timer mode
- (2) toggle of a port pin — High Speed Output mode
- (3) a reset — Watchdog Timer mode.

Examples of each compare mode will follow.

SOFTWARE TIMER

In most applications a software timer is used to trigger interrupt routines which must occur at periodic intervals. Figure 9 shows the sequence of events for the Software Timer mode. The user preloads a 16-bit value in a module's compare registers. When a match occurs between this compare value and the PCA timer, an event flag is set and an interrupt is flagged. An interrupt is then generated if it has been enabled.

If necessary, a new 16-bit compare value can be loaded into (CCAP0H, CCAP0L) during the interrupt routine. The user should be aware that the hardware temporarily disables the comparator function while these registers are being updated so that an invalid match will not occur. That is, a write to the low byte (CCAPn0) disables the comparator while a write to the high byte (CCAP0H) re-enables the comparator. For this reason, user software must write to CCAP0L first, then CCAP0H. The user may also want to hold off any interrupts from occurring while these registers are being updated. This can easily be done by clearing the EA bit. See the code example in Listing 5.

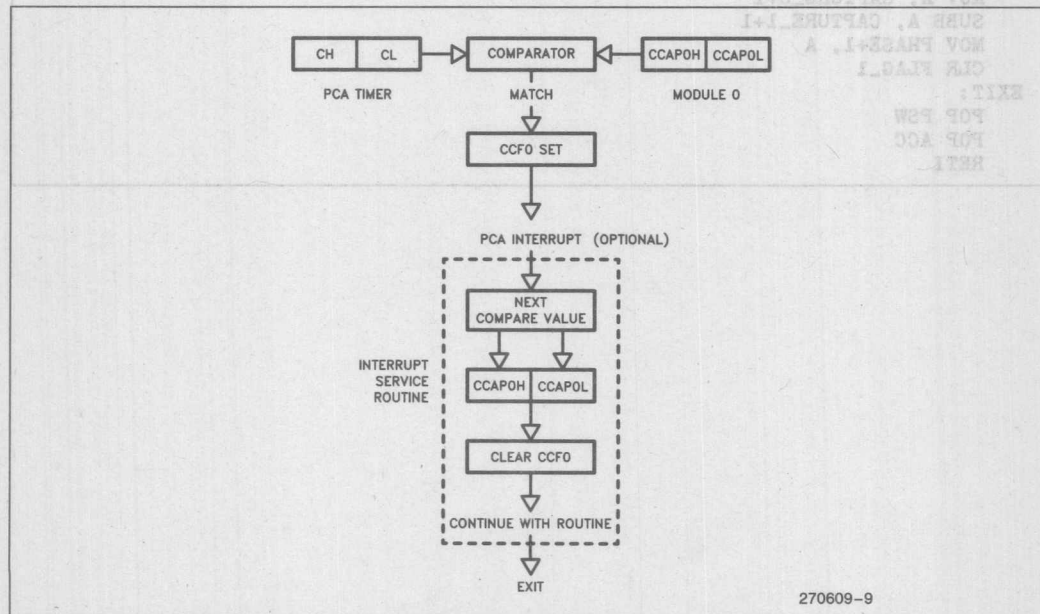


Figure 9. Software Timer Mode (Module 0)

Listing 5. Software Timer

```

; Generate an interrupt in software every 20 msec
;
; Frequency = 12 MHz
; PCA clock input = 1/12 x Fosc → 1 μsec
;
; Calculate reload value for compare registers:
;
;      20 msec
; ----- = 20,000 counts
; 1 μsec/count
;
;
; ORG 0000H
; JMP PCA_INIT
; ORG 0033H
; JMP PCA_INTERRUPT
;
; PCA_INIT:
; Initialize PCA timer same as in Listing 1
; MOV CCAPMO, #49H      ; Module 0 in Software Timer mode
; MOV CCAPOL, #LOW(20000) ; Write to low byte first
; MOV CCAPH, #HIGH(20000)
;
; SETB EC                ; Enable PCA interrupt
; SETB EA                ; Turn on PCA timer
; SETB CR
;
; *****
; Main program goes here
; *****
;
; PCA_INTERRUPT:
; CLR CCFO                ; Clear Module 0's event flag
; PUSH ACC
; PUSH PSW
; CLR EA                  ; Hold off interrupts
; MOV A, #LOW(20000)      ; 16-Bit Add
; ADD A, CCAPOL            ; Next match will occur
; MOV CCAPOL, A           ; 20,000 counts later
; MOV A, #HIGH(20000)
; ADDC A, CCAPH
; MOV CCAPH, A
; SETB EA
;
;
; Continue with routine
;
; POP PSW
; POP ACC
; RETI

```

HIGH SPEED OUTPUT

The High Speed Output (HSO) mode toggles a port pin when a match occurs between the PCA timer and the pre-loaded value in the compare registers (see Figure 10). The HSO mode is more accurate than toggling pins in software because the toggle occurs *before* branching to an interrupt, i.e. interrupt latency will not effect the accuracy of the output. In fact, the interrupt is optional. Only if the user wants to change the time for the next toggle is it necessary to update the compare registers. Otherwise, the next toggle will occur when the PCA timer rolls over and matches the last compare value. Examples of both are shown.

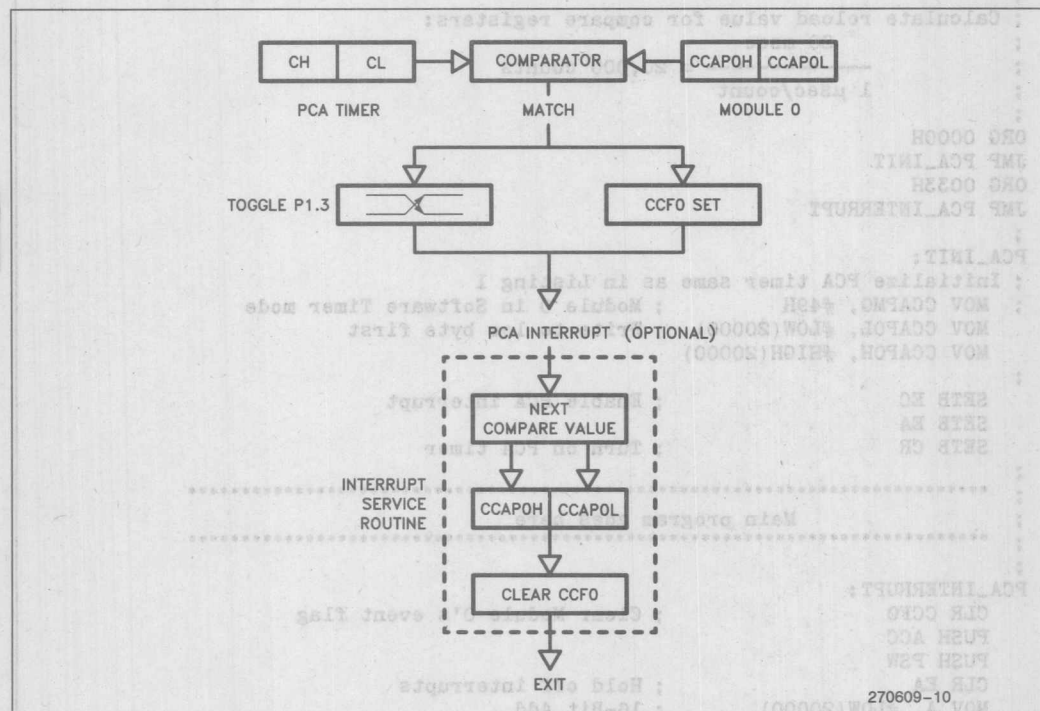


Figure 10. High Speed Output Mode (Module 0)

Without any CPU intervention, the fastest waveform the PCA can generate with the HSO mode is a 30.5 Hz signal at 16 MHz. Refer to Listing 6. By changing the PCA clock input, slower waveforms can also be generated.

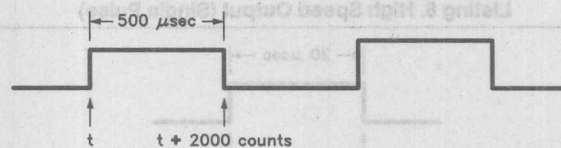
Listing 6. High Speed Output (Without Interrupt)

```

; Maximum output with HSO mode without interrupts = 30.5 Hz signal
; Frequency = 16 MHz
; PCA clock input = 1/4 x Fosc -> 250 nsec
;
MOV CMOD, #02H
MOV CL, #00H
MOV CH, #00H
MOV CCAPMO, #4CH ; HSO mode without interrupt enabled
MOV CCAPOL, #0FFH ; Write to low byte first
MOV CCAP0H, #0FFH ; P1.3 will toggle every 216 counts
; or 16.4 msec
; Period = 30.5 Hz
; Turn on PCA timer
SETB CR
  
```


In this next example, the PCA interrupt is used to change the compare value for each toggle. This way a variable frequency output can be generated. Listing 7 shows an output of 1 KHz at 16 Mhz.

Listing 7. High Speed Output (With Interrupt)



$$\frac{500 \mu\text{sec}}{250 \text{ nsec/count}} = 2000 \text{ counts}$$

270609-11

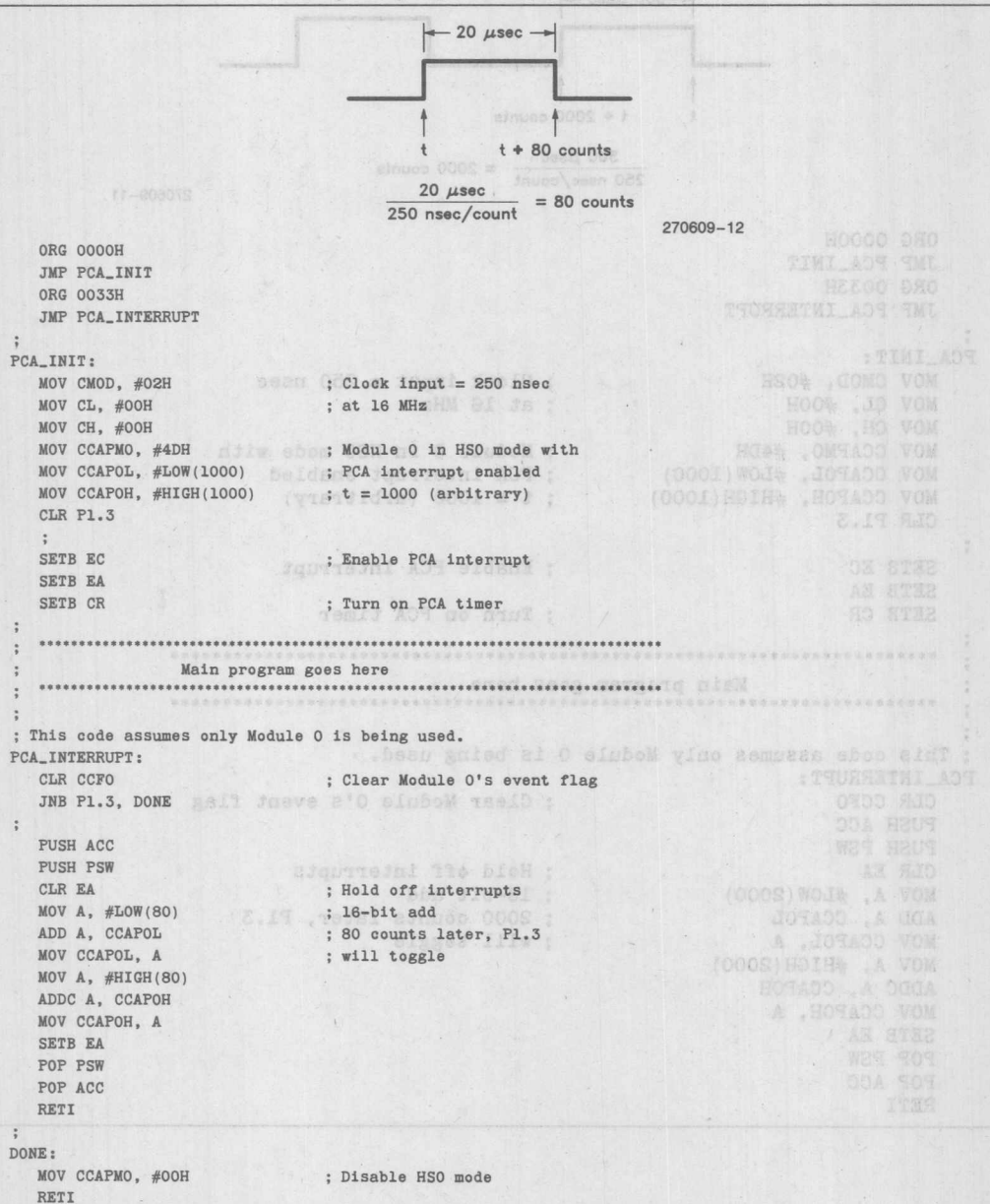
```

ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT
;
PCA_INIT:
MOV CMOD, #02H          ; Clock input = 250 nsec
MOV CL, #00H            ; at 16 MHz
MOV CH, #00H
MOV CCAPMO, #4DH        ; Module 0 in HSO mode with
MOV CCAPOL, #LOW(1000)   ; PCA interrupt enabled
MOV CCAPOH, #HIGH(1000) ; t = 1000 (arbitrary)
CLR P1.3
;
SETB EC                 ; Enable PCA interrupt
SETB EA
SETB CR                 ; Turn on PCA timer
;
; *****
; Main program goes here
; *****
;
; This code assumes only Module 0 is being used.
PCA_INTERRUPT:
CLR CCFO                ; Clear Module 0's event flag
PUSH ACC
PUSH PSW
CLR EA                  ; Hold off interrupts
MOV A, #LOW(2000)        ; 16-bit add
ADD A, CCAPOL            ; 2000 counts later, P1.3
MOV CCAPOL, A            ; will toggle
MOV A, #HIGH(2000)
ADDC A, CCAPOH
MOV CCAPOH, A
SETB EA
POP PSW
POP ACC
RETI

```

Another option with the HSO mode is to generate a single pulse. Listing 8 shows the code for an output with a pulse width of 20 μsec . As in the previous example, the PCA interrupt will be used to change the time for the toggle. The first toggle will occur at time "t". After 80 counts of the PCA timer, 20 μsec will have expired, and the next toggle will occur. Then the HSO mode will be disabled.

Listing 8. High Speed Output (Single Pulse)



WATCHDOG TIMER

An on-board watchdog timer is available with the PCA to improve the reliability of the system without increasing chip count. Watchdog timers are useful for systems which are susceptible to noise, power glitches, or electrostatic discharge. Module 4 is the only PCA module which can be programmed as a watchdog. However, this module can still be used for other modes if the watchdog is not needed.

Figure 11 shows a diagram of how the watchdog works. The user pre-loads a 16-bit value in the compare registers. Just like the other compare modes, this 16-bit value is compared to the PCA timer value. If a match is allowed to occur, an internal reset will be generated. This will not cause the RST pin to be driven high.

In order to hold off the reset, the user has three options:

- (1) periodically change the compare value so it will never match the PCA timer,
- (2) periodically change the PCA timer value so it will never match the compare value, or
- (3) disable the watchdog by clearing the WDTE bit before a match occurs and then re-enable it.

The first two options are more reliable because the watchdog timer is never disabled as in option #3. If the program counter ever goes astray, a match will eventually occur and cause an internal reset. The second option is also not recommended if other PCA modules are being used. Remember, the PCA timer is the time base for *all* modules; changing the time base for other modules would not be a good idea. Thus, in most applications the first solution is the best option.

Listing 9 shows the code for initializing the watchdog timer. Module 4 can be configured in either compare mode, and the WDTE bit in CMOD must also be set. The user's software then must periodically change (CCAP4H,CCAP4L) to keep a match from occurring with the PCA timer (CH,CL). This code is given in the WATCHDOG routine.

This routine should not be part of an interrupt service routine. Why? Because if the program counter goes astray and gets stuck in an infinite loop, interrupts will still be serviced and the watchdog will keep getting reset. Thus, the purpose of the watchdog would be defeated. Instead call this subroutine from the main program within 2^{16} count of the PCA timer.

2

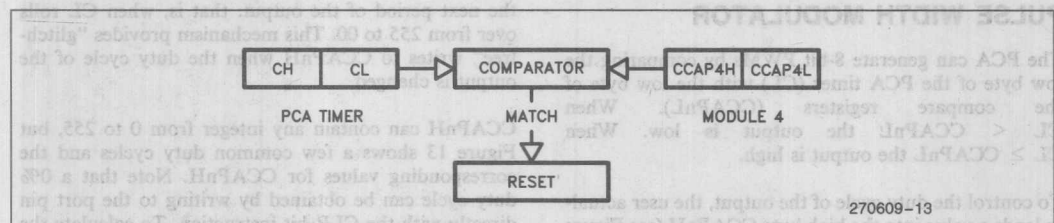


Figure 11. Watchdog Timer Mode (Module 4)

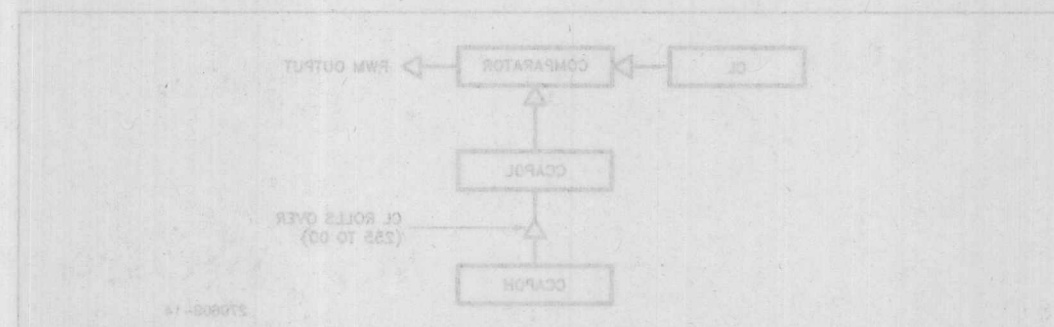


Figure 12. PWM Mode (Module 0)

Listing 9. Watchdog Timer

```

INIT_WATCHDOG:
    MOV CCAPM4, #4CH      ; Module 4 in compare mode
    MOV CCAP4L, #0FFH     ; Write to low byte first
    MOV CCAP4H, #0FFH     ; Before PCA timer counts up to
                          ; FFFF Hex, these compare values
                          ; must be changed
    ORL CMOD, #40H        ; Set the WDTE bit to enable the
                          ; watchdog timer without changing
                          ; the other bits in CMOD
    ;
    ; *****
    ; Main program goes here, but CALL WATCHDOG periodically.
    ; *****
    ;
WATCHDOG:
    CLR EA                ; Hold off interrupts
    MOV CCAP4L, #00       ; Next compare value is within
    MOV CCAP4H, CH         ; 255 counts of the current PCA
    SETB EA               ; timer value
    RET

```

PULSE WIDTH MODULATOR

The PCA can generate 8-bit PWMs by comparing the low byte of the PCA timer (CL) with the low byte of the compare registers (CCAPnL). When $CL < CCAPnL$ the output is low. When $CL \geq CCAPnL$ the output is high.

To control the duty cycle of the output, the user actually loads a value into the high byte CCAPnH (see Figure 12). Since a write to this register is asynchronous, a new value is not shifted into CCAPnL for comparison until

the next period of the output: that is, when CL rolls over from 255 to 00. This mechanism provides “glitch-free” writes to CCAPnH when the duty cycle of the output is changed.

CCAPnH can contain any integer from 0 to 255, but Figure 13 shows a few common duty cycles and the corresponding values for CCAPnH. Note that a 0% duty cycle can be obtained by writing to the port pin directly with the CLR bit instruction. To calculate the CCAPnH value for a given duty cycle, use the following equation:

$$CCAPnH = 256 (1 - \text{Duty Cycle})$$

where CCAPnH is an 8-bit integer and Duty Cycle is expressed as a fraction.

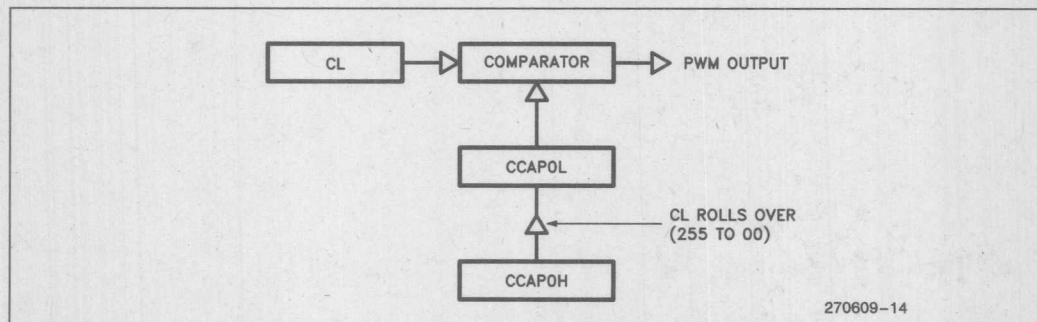


Figure 12. PWM Mode (Module 0)

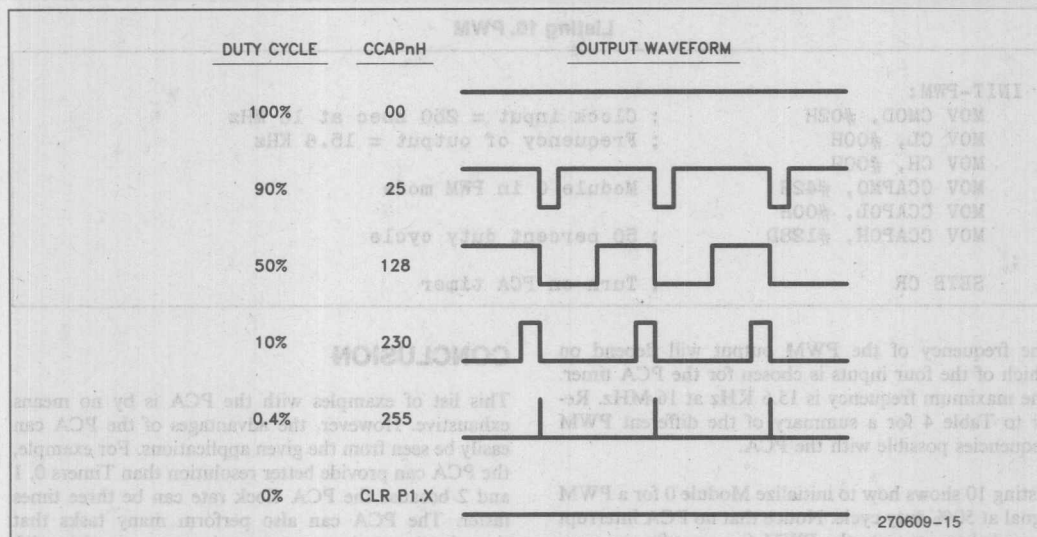


Figure 13. CCAPnH Varies Duty Cycle

Table 4. PWM Frequencies.

PCA Timer Mode	PWM Frequency	
	12 MHz	16 MHz
1/12 Osc. Frequency	3.9 KHz	5.2 KHz
1/4 Osc. Frequency	11.8 KHz	15.6 KHz
Timer 0 Overflow:		
8-bit	15.5 Hz	20.3 Hz
16-bit	0.06 Hz	0.08 Hz
8-bit Auto-Reload	3.9 KHz to 15.3 Hz	5.2 KHz to 20.3 Hz
External Input (Max)	5.9 KHz	7.8 KHz

Listing 10. PWM

```
INIT-PWM:
MOV CMOD, #02H           ; Clock input = 250 nsec at 16 MHz
MOV CL, #00H             ; Frequency of output = 15.6 KHz
MOV CH, #00H
MOV CCAPM0, #42H         ; Module 0 in PWM mode
MOV CCAPOL, #00H
MOV CCAPOH, #128D        ; 50 percent duty cycle
;
SETB CR                  ; Turn on PCA timer
```

The frequency of the PWM output will depend on which of the four inputs is chosen for the PCA timer. The maximum frequency is 15.6 KHz at 16 MHz. Refer to Table 4 for a summary of the different PWM frequencies possible with the PCA.

Listing 10 shows how to initialize Module 0 for a PWM signal at 50% duty cycle. Notice that no PCA interrupt is needed to generate the PWM (i.e no software overhead!). To create a PWM output on the 8051 requires a hardware timer plus software overhead to toggle the port pin. The advantage of the PCA is obvious, not to mention it can support up to 5 PWM outputs with just one chip.

CONCLUSION

This list of examples with the PCA is by no means exhaustive. However, the advantages of the PCA can easily be seen from the given applications. For example, the PCA can provide better resolution than Timers 0, 1 and 2 because the PCA clock rate can be three times faster. The PCA can also perform many tasks that these hardware timers can not, i.e. measure phase differences between signals or generate PWMs. In a sense, the PCA provides the user with five more timer/counters in addition to Timers 0, 1 and 2 on the 8XC51FA/FB.

Appendix A includes test routines for all the software examples in this application note. The divide routine for calculating duty cycles is in Appendix B. And finally, Appendix C is a table of the Special Function Registers for the 8XC51FA/FB with the new or modified registers **boldfaced**.

APPENDIX A TEST ROUTINES

Listing 1a - Measuring Pulse Widths

```

;
$nomod51
$nosymbols
$nolist
#include (reg252.pdf)
$list
;
; Variables
;
CAPTURE          DATA      30H
PULSE_WIDTH      DATA      32H
FLAG             BIT         20H.0
;
ORG 0000H
JMP PCA_INIT
;
ORG 0033H
JMP PCA_INTERRUPT
;
; Initialize PCA timer
PCA_INIT:  MOV CMOD, #00H      ; Input to PCA timer = 1/12 x Fosc
          MOV CH, #00
          MOV CL, #00
;
; Initialize Module 0 in capture mode
          MOV CCAPM0, #21H    ; Capture positive edge first on P1.3
          MOV CCAP0H, #00
          MOV CCAP0L, #00
;
          SETB EC              ; Enable PCA interrupt
          SETB EA              ; Turn PCA timer on
          SETB CR              ; Clear test flag
          CLR FLAG
;
;-----
; Test program only
;-----
WAIT:     JMP $                ; Wait for PCA interrupt
          JMP WAIT
;-----
; This code assumes Module 0 is the only module being used. If
; other PCA module's are being used, software must check which
; module's event flag caused the interrupt.
;
PCA_INTERRUPT:
          CLR CCF0             ; Clear module 0's event flag
          JB FLAG, SECOND_CAPTURE
;
FIRST_CAPTURE:
          MOV CAPTURE, CCAP0L
          MOV CAPTURE+1, CCAP0H

```

270609-16

MOV CCAPM0, #11H

; Change module to now capture

SETB FLAG

; falling edges

RETI

; Signify first capture complete

; SECOND_CAPTURE:

PUSH ACC

PUSH PSW

CLR C

MOV A, CCAP0L

; 16-bit subtract

SUBB A, CAPTURE

MOV PULSE_WIDTH, A

MOV A, CCAP0H

SUBB A, CAPTURE+1

MOV PULSE_WIDTH+1, A

MOV CCAPM0, #21H

; Optional if user wants to measure

CLR FLAG

; next pulse width

POP PSW

POP ACC

RETI

; END

Listing 1b - Measuring Periods

```

;
;
$nomod51
$nosymbols
$nolist
$include (reg252.pdf)
$list
;
;   Variables
;
CAPTURE      DATA      30H
PERIOD       DATA      32H
FLAG         BIT        20H.0
;
ORG 0000H
JMP PCA_INIT
;
ORG 0033H
JMP PCA_INTERRUPT
;
;   Initialize PCA timer
PCA_INIT:    MOV CMOD, #00H      ; Input to timer = 1/12 x Fosc
            MOV CH, #00H
            MOV CL, #00
;
;   Initialize Module 0 in capture mode
            MOV CCAPM0, #21H      ; Capture rising edges on P1.3
;
            MOV CCAP0H, #00
            MOV CCAP0L, #00
;
            SETB EC                ; Enable PCA interrupt
            SETB EA                ; Turn PCA timer on
            SETB CR                ; Clear test flag
            CLR FLAG
;
; *****
;   Test program only
;
WAIT:        JMP $                ; Wait for PCA interrupt
            JMP WAIT
;
; *****
;   This code assumes only Module 0 is being used. If other modules
;   are being used, software must check which module's flag caused
;   the interrupt.
;
PCA_INTERRUPT:
            CLR CCF0                ; Clear module 0's event flag
            JB FLAG, SECOND_CAPTURE
;
FIRST_CAPTURE:
            MOV CAPTURE, CCAP0L
            MOV CAPTURE+1, CCAP0H

```

270609-18

```

        SETB FLAG                ; Signify first capture complete
        RETI

; SECOND_CAPTURE:
        PUSH ACC
        PUSH PSW
        CLR C
        MOV A, CCAP0L           ; 16-Bit subtraction
        SUBB A, CAPTURE
        MOV PERIOD, A
        MOV A, CCAP0H
        SUBB A, CAPTURE+1
        MOV PERIOD+1, A

;
        CLR FLAG
        POP PSW
        POP ACC
        RETI

;
END

```

270609-19

int

```

; Signify first capture complete
MOV CAPTURE+1, CCAP0H
SETB FLAG
RETI

```

NEXT_CAPTURE:

```

DJNZ SAMPLE_COUNT, EXIT
PUSH ACC
PUSH PSW
CLR C
MOV A, CCAP0L
SUBB A, CAPTURE
MOV PERIOD, A
MOV A, CCAP0H
SUBB A, CAPTURE+1
MOV PERIOD+1, A

```

; 16-Bit subtraction

```

; Reload for next capture
MOV SAMPLE_COUNT, #10D
CLR FLAG
POP PSW
POP ACC
RETI

```

EXIT:
;
END

270609-21

270609-22

FIRST_CAPTURE:

```

MOV CAPTURE, CCAP0L
MOV CAPTURE+1, CCAP0H
SETB FLAG_1
MOV CCAPM0, #31H
RET

```

; Signify first capture complete
; Capture either edge now

SECOND_CAPTURE:

```

PUSH ACC
PUSH PSW
JB FLAG_2, THIRD_CAPTURE
CLR C
MOV A, CCAP0L
SUBB A, CAPTURE
MOV PULSE_WIDTH, A
MOV A, CCAP0H
SUBB A, CAPTURE+1
MOV PULSE_WIDTH+1, A

;

SETB FLAG_2
POP PSW
POP ACC
RET

```

; Calculate pulse width
; 16-bit subtract

; Signify second capture complete

THIRD_CAPTURE:

```

CLR C
MOV A, CCAP0L
SUBB A, CAPTURE
MOV PERIOD, A
MOV A, CCAP0H
SUBB A, CAPTURE+1
MOV PERIOD+1, A

;

MOV CCAPM0, #21H
CLR FLAG_1
CLR FLAG_2
POP PSW
POP ACC
RET

```

; Calculate period
; 16-bit subtract

; Optional- reconfigure module to
; capture positive edges for
; next cycle

; END

270609-23

Listing 4 - Measuring Phase Differences

```

;
;
;nomod51
;nosymbols
;nolist
#include (reg252.pdf)
$!list
;
; Variables
;
CAPTURE_0      DATA      30H
CAPTURE_1      DATA      32H
PHASE          DATA      34H
;
FLAG_0         BIT        20H.0
FLAG_1         BIT        20H.1
;
ORG 0000H
JMP PCA_INIT
;
ORG 0033H
JMP PCA_INTERRUPT
;
; Initialize PCA timer
PCA_INIT:      MOV CMOD, #00H      ; Input to PCA timer = 1/12 x Fosc
              MOV CH, #00
              MOV CL, #00
;
; Initialize Modules 0 & 1 in capture mode
              MOV CCAPM0, #21H      ; Capture positive edges on P1.3
              MOV CCAP0H, #00
              MOV CCAP0L, #00
;
              MOV CCAPM1, #21H      ; Capture positive edges on P1.4
              MOV CCAP1H, #00
              MOV CCAP1L, #00
;
              MOV R0, #0FFH          ; Used for test program only
              MOV R1, #0FFH
;
              CLR FLAG_0             ; Clear test flags
              CLR FLAG_1
;
              SETB EC                ; Enable PCA interrupt
              SETB EA
              SETB CR                ; Turn PCA timer on
;

```

270609-24

```

; Test program only

MAIN:      CALL TOG1      ; Generate two waveforms
           CALL DELAY2    ; with known phase difference
           CALL TOG2
           JMP MAIN

TOG1:      CPL P1.6        ; These two waveforms are input to
           CALL DELAY1    ; P1.3 and P1.4
           RET

TOG2:      CPL P1.5
           CALL DELAY1
           RET

DELAY1:    DJNZ R0, $
           RET

DELAY2:    DJNZ R1, $
           RET

; *****
; This code assumes only Modules 0 and 1 are being used.
; *****

PCA_INTERRUPT:
           JB CCF0, MODULE_0      ; Determine which module's event
           JB CCF1, MODULE_1      ; caused the interrupt

MODULE_0:
           CLR CCF0              ; Clear Module 0's event flag
           MOV CAPTURE_0, CCAP0L
           MOV CAPTURE_0+1, CCAP0H
           JB FLAG_1, CALCULATE_PHASE ; If capture is complete on Module 1,
           ; go to calculation
           SETB FLAG_0          ; Signify capture complete on
           RETI                 ; Module 0

MODULE_1:
           CLR CCF1              ; Clear Module 1's event flag
           MOV CAPTURE_1, CCAP1L
           MOV CAPTURE_1+1, CCAP1H
           JB FLAG_0, CALCULATE_PHASE ; If capture is complete on Module 0,
           ; go to calculation
           SETB FLAG_1          ; Signify capture complete
           RETI                 ; Module 1

CALCULATE_PHASE:
           PUSH ACC              ; This calculation does not have to
           PUSH PSW              ; be completed in the interrupt
           CLR C                 ; service routine

           JB FLAG_0, MOD0_LEADING

```

270609-25

JB FLAG_1, MOD1_LEADING				
; MOD0_LEADING:				
MOV A, CAPTURE_1		; 16-bit subtraction		
SUBB A, CAPTURE_0				
MOV PHASE, A				
MOV A, CAPTURE_1+1				
SUBB A, CAPTURE_0+1				
MOV PHASE+1, A				
CLR FLAG_0				
JMP EXIT				
; MOD1_LEADING:				
MOV A, CAPTURE_0		; 16-bit subtraction		
SUBB A, CAPTURE_1				
MOV PHASE, A				
MOV A, CAPTURE_0+1				
SUBB A, CAPTURE_1+1				
MOV PHASE+1, A				
CLR FLAG_1				
EXIT:				
POP PSW				
POP ACC				
RETI				
; END				
			270609-26	

Listing 5. Software Timer

```

;
;nomod51
;nosymbols
;nolist
#include (reg252.pdf)
;list
; Software Timer mode which interrupts every 20 msec with Fosc = 12 MHz.
;
ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT
;
; Initialize PCA timer
PCA_INIT: MOV CMOD, #00H ; Input to PCA timer = 1/12 x Fosc
          MOV CH, #00
          MOV CL, #00
;
          MOV CCAPM0, #49H ; Software Timer mode with interrupt
          MOV CCAP0L, #LOW(20000) ; Write to low byte first
          MOV CCAP0H, #HIGH(20000)
;
          SETB EC ; Enable PCA interrupt
          SETB EA ; Turn PCA timer on
          SETB CR
;
;-----
; Test program only
;
WAIT:     JMP $ ; Wait for PCA interrupt
          JMP WAIT
;-----
;
; This code assumes Module 0 is the only module being used. If
; other PCA module's are being used, software must check which
; module's event flag caused the interrupt.
;
PCA_INTERRUPT:
          CLR CCF0 ; Clear module 0's event flag
          PUSH ACC
          PUSH PSW
          CLR EA ; Hold off interrupts
          MOV A, #LOW(20000) ; 16-bit add
          ADD A, CCAP0L ; Next match will occur 20,000
          MOV CCAP0L, A ; counts later
          MOV A, #HIGH(20000)
          ADDC A, CCAP0H
          MOV CCAP0H, A
          SETB EA
          POP PSW
          POP ACC
          RETI
END

```

270609-27

Listing 6. High Speed Output (without interrupt)

```

;
$nomod51
$nosymbols
$noist
$include (reg252.pdf)
$list
;
;
; HSO mode without PCA interrupt. Maximum frequency output = 30.5 Hz
; at Fosc = 16 MHz.
;
ORG 0000H
JMP PCA_INIT
;
; Initialize PCA timer
PCA_INIT: MOV CMOD, #02H      ; Input to PCA timer = 1/4 x Fosc
          MOV CH, #00
          MOV CL, #00
;
          MOV CCAPM0, #4CH    ; HSO Mode without interrupt enabled
          MOV CCAP0L, #0FFH   ; Write to low byte first
          MOV CCAP0H, #0FFH   ; P1.3 will toggle every 65,536 counts
                                ; or 16.4 msec at Fosc = 16 MHz
                                ; Period = 30.5 Hz
                                ; Turn PCA timer on
          SETB CR
;
END

```

2

```

; Listing 7. High Speed Output (with interrupts)

$nomod51
$nosymbols
$nolist
$include (reg252.pdf)
$list

;
; HSO mode with variable frequency. This example outputs a 1KHz signal
; with Fosc = 16 MHz.
;
ORG 0000H
JMP PCA_INIT
;
ORG 0033H
JMP PCA_INTERRUPT
;
; Initialize PCA timer
PCA_INIT:  MOV CMOD, #02H      ; Input to PCA timer = 1/4 x Fosc
           MOV CH, #00
           MOV CL, #00
;
           MOV CCAPM0, #4DH    ; HSO mode with interrupt enabled
           MOV CCAP0L, #LOW(1000) ; t = 1000 arbitrary
           MOV CCAP0H, #HIGH(1000)
           CLR P1.3
;
           SETB EC             ; Enable PCA interrupt
           SETB EA
           SETB CR             ; Turn PCA timer on
;
; *****
; Test program only
;
WAIT:      JMP $               ; Wait for PCA interrupt
           JMP WAIT
; *****

; This code assumes Module 0 is the only module being used. If
; other PCA module's are being used, software must check which
; module's event flag caused the interrupt.
;
PCA_INTERRUPT:
           CLR CCF0            ; Clear module 0's event flag
           PUSH ACC
           PUSH PSW
           CLR EA              ; Hold off interrupts
           MOV A, #LOW(2000)   ; 16-bit add
           ADD A, CCAP0L       ; 2000 counts later P1.3
           MOV CCAP0L, A       ; will toggle
           MOV A, #HIGH(2000)
           ADDC A, CCAP0H
           MOV CCAP0H, A
;
           SETB EA
           POP PSW
           POP ACC
           RETI
;
END

```

270609-29

270609-30

Listing 8. High Speed Output (Single Pulse)

```

;
; $nomod51
; $nosymbols
; $nolist
; $include (reg252.pdf)
; $list
;
; HSO mode generates a single pulse width of 20 usecs with Fosc = 16 MHz.
;
ORG 0000H
JMP PCA_INIT
;
ORG 0033H
JMP PCA_INTERRUPT
;
; Initialize PCA timer
PCA_INIT:  MOV CMOD, #02H      ; Input to PCA timer = 1/4 x Fosc
           MOV CH, #00
           MOV CL, #00
           MOV CCAPM0, #4DH    ; HSO mode with interrupt enabled
           MOV CCAP0L, #LOW(1000) ; t = 1000 arbitrary
           MOV CCAP0H, #HIGH(1000)
           CLR P1.3
;
           SETB EC              ; Enable PCA interrupt
           SETB EA              ; Delay for output to occur
           SETB CR              ; Turn PCA timer on
;
;-----
; Test program only
;
WAIT:     JMP $                ; Wait for PCA interrupt
           JMP WAIT
;-----
; This code assumes Module 0 is the only module being used. If
; other PCA module's are being used, software must check which
; module's event flag caused the interrupt.
PCA_INTERRUPT:
           CLR CCF0             ; Clear module 0's event flag
           JNB P1.3, DONE
;
           PUSH ACC
           PUSH PSW
           CLR EA               ; Hold off interrupts
           MOV A, #LOW(80)      ; 16-bit add
           ADD A, CCAP0L        ; 80 counts later P1.3
           MOV CCAP0L, A        ; will toggle
           MOV A, #HIGH(80)
           ADDC A, CCAP0H
           MOV CCAP0H, A
           SETB EA
           POP PSW
           POP ACC
           RETI
;
DONE:     MOV CCAPM0, #00H      ; Disable HSO mode
           RETI
END

```

270609-31

270609-32

Listing 10. Pulse Width Modulator

```

;
$nomod51
$nosymbols
$nolist
$include (reg252.pdf)
$list
;
;
; PWM mode -- Maximum frequency output = 15.6 KHz with Fosc = 16 Mhz.
;
ORG 0000H
JMP PCA_INIT
;
; Initialize PCA timer
PCA_INIT:  MOV CMOD, #02H      ; Input to PCA timer = 1/4 x Fosc
           MOV CH, #00        ; At 16 MHz, frequency = 15.6 KHz
           MOV CL, #00
;
           MOV CCAPM0, #42H    ; PWM Mode
           MOV CCAP0L, #00H    ; Write to low byte first
           MOV CCAP0H, #128D   ; 50 percent duty cycle
           SETB CR             ; Turn PCA timer on
;
END

```

APPENDIX B

Duty Cycle Calculation

\$DEBUG

SHORT_DIVISION SEGMENT CODE

EXTRN DATA(PULSE_WIDTH, PERIOD, DUTY_CYCLE)
PUBLIC DUTY_CYCLE_CALCULATION

RSEG SHORT_DIVISION

```

;
; *****
;          DUTY_CYCLE_CALCULATION
; *****
;
;  CALCULATES DUTY_CYCLE = PULSE_WIDTH / PERIOD
;

```

Inputs to this routine are 16-bit pulse width and period measurements of a rectangular waveform. The output is a 9-bit BCD number representing the duty cycle of the waveform. The low 8 bits of the result are returned in DUTY_CYCLE. The 9th bit is the carry bit in the PSW. If the duty cycle is between 0 and 99 percent, the carry bit is 0 and DUTY_CYCLE contains the two BCD digits representing the duty cycle as a percent. If the duty cycle is 100 percent, the carry bit is 1 and DUTY_CYCLE contains 0.

INPUTS: PULSE_WIDTH 2 bytes in externally defined DATA
(low byte at PULSE_WIDTH, high byte at PULSE_WIDTH+1)

PERIOD 2 bytes in externally defined DATA
(low byte at PERIOD, high byte at PERIOD+1)

OUTPUT: DUTY_CYCLE 1 byte in externally defined DATA

VARIABLES AND REGISTERS MODIFIED:

PULSE_WIDTH, DUTY_CYCLE
ACC, B, PSW, R2, R3

ERROR EXIT: Exit with OV = 1 indicates PULSE_WIDTH > PERIOD.

```

; *****
;
; DUTY_CYCLE_CALCULATION:
;   MOV  A,PERIOD+1
;   CJNE A,PULSE_WIDTH+1,NOT_EQUAL
;   MOV  A,PERIOD
;   CJNE A,PULSE_WIDTH,NOT_EQUAL
;

```

270609-35

2

FINAL_TIMES_TWO:

```

MOV A,PULSE_WIDTH
RLC A
MOV PULSE_WIDTH,A
MOV A,PULSE_WIDTH+1
RLC A
MOV PULSE_WIDTH+1,A
MOV A,R3
RLC A
MOV R3,A
FINAL_COMPARE:
CJNE R3,#0,FINAL_DONE
MOV A,PULSE_WIDTH+1
CJNE A,PERIOD+1,FINAL_DONE
MOV A,PULSE_WIDTH
CJNE A,PERIOD,FINAL_DONE
FINAL_DONE:
JC CONVERT_TO_BCD
MOV A,DUTY_CYCLE
ADD A,#1
MOV DUTY_CYCLE,A
JNC CONVERT_TO_BCD
CLR OV
RET

```

CONVERT_TO_BCD:

```

MOV A,DUTY_CYCLE
MOV B,#10
MUL AB
XCH A,B
SWAP A
MOV DUTY_CYCLE,A
MOV A,#10
MUL AB
XCH A,B
ORL DUTY_CYCLE,A
MOV A,#10
MUL AB
MOV A,B
CJNE A,#5,TEST
TEST: JBC CY,OUT
MOV A,DUTY_CYCLE
ADD A,#1
DA A
MOV DUTY_CYCLE,A
OUT: RET

```

END

270609-37

APPENDIX C

A map of the Special Function Register (SFR) space is shown in Table A1. Those registers which are new or have new bits added for the 83C51FA and 83C51FB have been **boldfaced**.

Note that not all of the addresses are occupied. Unoccupied addresses are not implemented on the chip.

Read accesses to these addresses will in general return random data, and write accesses will have no effect.

User software should not write 1s to these unimplemented locations, since they may be used in future 8051 family products to invoke new features. In that case the reset or inactive values of the new bits will always be 0, and their active values will be 1.

Table A1. Special Function Register Memory Map and Values After Reset

F8		CH 00000000	CCAP0H XXXXXXXX	CCAP1H XXXXXXXX	CCAP2H XXXXXXXX	CCAP3H XXXXXXXX	CCAP4H XXXXXXXX	FF
F0	* B 00000000							F7
E8		CL 00000000	CCAP0L XXXXXXXX	CCAP1L XXXXXXXX	CCAP2L XXXXXXXX	CCAP3L XXXXXXXX	CCAP4L XXXXXXXX	EF
E0	* ACC 00000000							E7
D8	CCON 00X00000	CMOD 00XXX000	CCAPM0 X0000000	CCAPM1 X0000000	CCAPM2 X0000000	CCAPM3 X0000000	CCAPM4 X0000000	DF
D0	* PSW 00000000							D7
C8	T2CON 00000000	T2MOD XXXXXXXX0	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000		CF
C0								C7
B8	* IP X0000000	SADEN 00000000						BF
B0	* P3 11111111							B7
A8	* IE 00000000	SADDR 00000000						AF
A0	* P2 11111111							A7
98	* SCON 00000000	* SBUF XXXXXXXX						9F
90	* P1 11111111							97
88	* TCON 00000000	* TMOD 00000000	* TL0 00000000	* TL1 00000000	* TH0 00000000	* TH1 00000000		8F
80	* P0 11111111	* SP 00000111	* DPL 00000000	* DPH 00000000			*PCON ** 00XX0000	87

* = Found in the 8051 core (See 8051 Hardware Description in the Embedded Controller Handbook for explanations of these SFRs).

** = See description of PCON SFR. Bit PCON.4 is not affected by reset.

X = Undefined.

APPENDIX C

Read accesses to these addresses will in general return random data, and write accesses will have no effect.

User software should not write to these unimplemented locations, since they may be used in future 8051 family products to invoke new features. In that case the test or inactive value of the new bits will always be 0, and their active value will be 1.

A map of the Special Function Register (SFR) space is shown in Table A1. Those registers which are new or have new bits added for the 83C128A and 83C128B have been highlighted.

Note that not all of the addresses are occupied. Unoccupied addresses are not implemented on the chip.

September 1988

FF	CH	00000000	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX
F7		00000000																								
E7	CL	00000000	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX
D7	ACC	00000000																								
C7	CON	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	00X00000	
B7	PSW	00000000																								
A7	TCON	00000000	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX
97																										
87	IP	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX
77	PS	11111111																								
67	IE	00000000	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX
57	PS	11111111																								
47	SCON	00000000	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX
37	PI	11111111																								
27	TCON	00000000	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX
17	TH1	00000000																								
07	PCON	00X00000																								

Small DC Motor Control

JAFAR MODARES
ECO APPLICATIONS

JAFAR MODARES
ECO APPLICATIONS

Order Number: 270622-001

SMALL DC MOTOR CONTROL

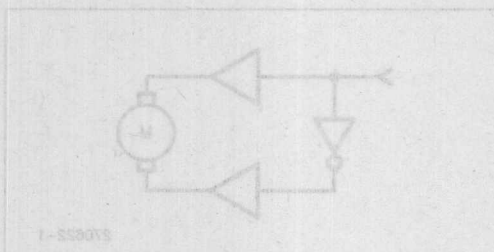


Figure 1. Reversible Motor Driver Circuit

Varying the speed requires changing the voltage level of the input to the motor, and that means changing the input level to the motor driver. In a digitally-controlled system, the analog signal to the driver must come from some form of D/A converter. But adding a D/A converter to the circuit adds to the chip count, which means more cost, higher power consumption, and reduced reliability of the system.

The other alternative is to vary the pulse width of a digital signal input to the motor. By varying the pulse width, the average voltage delivered to the motor changes and so does the speed of the motor. A digital circuit that does this is called a Pulse Width Modulator (PWM). The 83C51FA can be configured to have up to 2 on-board pulse width modulators.

THE 83C51FA

The 83C51FA is an 8-bit microcontroller based on the 8051 architecture. It is an enhanced version of the 83C51 and incorporates many new features including the Programmable Counter Array (PCA).

Included in the Programmable Counter Array is a 16-bit free-running timer and 2 separate modules.

The PCA timer has two 8-bit registers called CL (low byte) and CH (high byte), and is shared by all modules. It can be programmed to take input from four different sources. The inputs provide flexibility in choosing the count rate of the timer. The maximum count rate is 4 MHz ($\frac{1}{2}$ of the oscillator frequency).

Some of the port I pins are used to interface each module and the timer to the outside world. When the port pins are not used by the PCA modules, they may be used as regular I/O pins.

The modules of the PCA can be programmed to perform in one of the following modes: capture mode

CONTENTS

	PAGE
INTRODUCTION	2-292
DC MOTORS	2-292
THE 83C51FA	2-292
SETTING UP THE PCA	2-293
HARDWARE REQUIREMENTS	2-294
DRIVER CIRCUIT	2-295
NOISE CONSIDERATIONS	2-295
OPEN LOOP AND CLOSED LOOP SYSTEMS	2-296
FEEDBACK	2-296
SOFTWARE/CPU OVERHEAD	2-298
ELECTRICAL BRAKING	2-299
STEPPING A DC MOTOR	2-300
TIME DELAYS	2-300
CONCLUSION	2-302
APPENDIX	2-303

However, some motor applications do require precise positioning. Examples are high resolution plotters, printers, disk drives, robots, etc. Stepper motors are frequently used in these applications. There are also applications which require precise speed control along with some position accuracy. Video recorders, compact disk drives, high capacity cassette recorders are examples of this category.

By controlling DC motors accurately, they can overlap many applications of stepper motors. The cost of the control system depends on the accuracy of the encoder and the speed of the processor.

The 83C51FA can control a DC motor accurately with minimum hardware at a very low cost. The microcontroller, as the brain of a system, can digitally control the angular velocity of the motor by monitoring the feedback lines and driving the output lines. In addition, it can perform other tasks which may be needed in the application.

Almost every application that uses a DC motor requires it to reverse its direction of rotation or vary its speed. Reversing the direction is simply done by changing

INTRODUCTION

This application note shows how an 83C51FA can be used to efficiently control DC motors with minimum hardware requirements. It also discusses software implementation and presents helpful techniques as well as sample code needed to realize precision control of a motor.

There is also a brief overview of the new features of the 83C51FA. This new feature is called the Programmable Counter Array (PCA) and is capable of delivering Pulse Width Modulated signals (PWM) through designated I/O pins.

It is assumed that the reader is familiar with the MCS-51 architecture and its assembly language. For more information about the 8051 architecture and the PCA refer to the Embedded Controller Handbook Volume 1 (order no. 210918-006).

This document will not discuss stepper motors or motor control algorithms.

DC MOTORS

DC motors are widely used in industrial and consumer applications. In many cases, absolute precision in movement is not an issue, but precise speed control is. For example, a DC motor in a cassette player is expected to run at a constant speed. It does not have to run for precise increments which are fractions of a turn and stop exactly at a certain point.

However, some motor applications do require precise positioning. Examples are high resolution plotters, printers, disk drives, robotics, etc. Stepper motors are frequently used in those applications. There are also applications which require precise speed control along with some position accuracy. Video recorders, compact disk drives, high quality cassette recorders are examples of this category.

By controlling DC motors accurately, they can overlap many applications of stepper motors. The cost of the control system depends on the accuracy of the encoder and the speed of the processor.

The 83C51FA can control a DC motor accurately with minimum hardware at a very low cost. The microcontroller, as the brain of a system, can digitally control the angular velocity of the motor, by monitoring the feedback lines and driving the output lines. In addition it can perform other tasks which may be needed in the application.

Almost every application that uses a DC motor requires it to reverse its direction of rotation or vary its speed. Reversing the direction is simply done by chang-

ing the polarity of the voltage applied to the motor. Figure 1 shows a simplified symbolic representation of a driver circuit which is capable of reversing the polarity of the input to the motor.

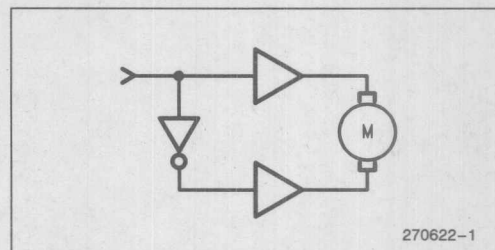


Figure 1. Reversible Motor Driver Circuit

Varying the speed requires changing the voltage level of the input to the motor, and that means changing the input level to the motor driver. In a digitally-controlled system, the analog signal to the driver must come from some form of D/A converter. But adding a D/A converter to the circuit adds to the chip count, which means more cost, higher power consumption, and reduced reliability of the system.

The other alternative is to vary the pulse width of a digital signal input to the motor. By varying the pulse width the average voltage delivered to the motor changes and so does the speed of the motor. A digital circuit that does this is called a Pulse Width Modulator (PWM). The 83C51FA can be configured to have up to 5 on-board pulse width modulators.

THE 83C51FA

The 83C51FA is an 8-bit microcontroller based on the 8051 architecture. It is an enhanced version of the 87C51 and incorporates many new features including the Programmable Counter Array (PCA).

Included in the Programmable Counter Array is a 16-bit free running timer and 5 separate modules.

The PCA timer has two 8-bit registers called CL (low byte) and CH (high byte), and is shared by all modules. It can be programmed to take input from four different sources. The inputs provide flexibility in choosing the count rate of the timer. The maximum count rate is 4 MHz ($1/4$ of the oscillator frequency).

Some of the port 1 pins are used to interface each module and the timer to the outside world. When the port pins are not used by the PCA modules, they may be used as regular I/O pins.

The modules of the PCA can be programmed to perform in one of the following modes: capture mode,

compare mode, high speed output mode, pulse width modulator (PWM) mode, or watchdog timer mode (only module 4).

Every module has an 8-bit mode register called CCAPMn (Figure 2), and a 16-bit compare/capture register called CCAPnL & CCAPnH, where n can be any value from 0 to 4 inclusive. By setting the appropriate bits in the mode register you can program each module to operate in one of the aforementioned modes.

7	6	5	4	3	2	1	0
—	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn

CCAPMn

ECOMn — Enables the comparator function. Must be set for functions which require comparing of the compare/capture registers with the 16-bit timer, i.e., software timer, high-speed output, watchdog timer, and PWM.

CAPPn — Capture on positive edge of signal.

CAPNn — Capture on negative edge of signal.

MATn — Find a match between the capture/compare and 16-bit timer.

TOGn — Toggle I/O pin upon a match between capture/compare registers and 16-bit timer.

PWMn — Generate PWM on I/O pin upon a match between the low byte of capture/compare and the low byte of PCA timer.

ECCFn — Enables compare/capture flag CCFn in the CCON register to generate an interrupt.

Figure 2. CCAPMn Register

When a module is programmed in capture mode, an external signal on the corresponding port pin will cause a capture of the current value of the 16-bit timer. By setting bits CAPPn or CAPNn or both, the module can be programmed to capture on the rising edge, falling edge, or either edge of the signal. If enabled, an interrupt is generated at the time of capture.

When module is to perform in one of the compare modes (software timer, high speed output, watch dog timer, PWM), the user loads the capture/compare registers with a calculated value, which is compared to the contents of the 16-bit timer, and causes an event as soon as the values match. It can also generate an interrupt.

PWM is one of the compare modes and is the only one which uses only 8 bits of the capture/compare register. The user writes a value (0 to FFH) into the high byte (CCAPnH) of the selected module. This value is transferred into the lower byte of the same module and is compared to the low byte of the PCA timer. While $CL < CCAPnL$ the output on the corresponding pin is a logic 0. When $CL > CCAPnL$, the output is a logic 1.

In this application note we will see how a module can be programmed to perform as a PWM to control the speed and direction of a DC motor.

SETTING UP THE PCA

The 83C51FA has several Special Function Registers (SFRs) that are unknown to ASM51 versions before 2.4. The names of these SFRs must be defined by DATA directive or be defined in a separate file and be included at the time of compilation. Such a file has already been created and is included in the ASM51 package version 2.4.

Two special function registers are dedicated to the PCA timer to allow mode selection and control of the timer. These registers are CCON and CMOD and are shown in figure 3. CCON contains the PCA timer ON/OFF bit (CR), timer rollover flag (CF) and module flags (CCFn). Module flags are used to determine which module causes the PCA interrupt.

7	6	5	4	3	2	1	0
CF	CR	—	CCF4	CCF3	CCF2	CCF1	CCF0

Address 0D8H
Bit Addressable
Reset Value = 00X0 0000 B

CCON

7	6	5	4	3	2	1	0
CIDL	WDTE	—	—	—	CPS1	CPS0	ECF

Address 0D9H
Not Bit Addressable
Reset Value = 00XX X000 B

CMOD

Figure 3. CCON and CMOD Registers

First the clock source for the PCA timer must be defined. The 16 bit timer may have one of four sources for its input. These sources are: osc freq/4, osc freq/12, timer 0 overflow, and external clock.

Two bits in the CMOD register are dedicated to selecting one of the sources for the PCA timer input. They are bits 1 and 2 of CMOD which are called CPS0 and CPS1. CMOD is not bit addressable, thus the value

must be loaded as a byte. Figure 4 shows all the sources and the corresponding values of CPS0 and CPS1.

CPS1	CPS0	TIMER INPUT SOURCE
0	0	Internal clock, Fosc/12
0	1	Internal clock, Fosc/4
1	0	Timer 0 overflow
1	1	External clock (input on P1.2)

Figure 4. Timer Input Source

Next the appropriate module must be programmed as a PWM. As it was noted earlier, the 8-bit mode register for each module is called CCAPMn (see figure 2). Bit 1 of each register is called PWMn. This bit along with ECOMn (bit 6 of the same register) must be set to program the module in the PWM mode. PWM is one of the compare functions of the PCA, and ECOMn enables the compare function. Thus, the hex value that must be loaded into the appropriate CCAPMn register is 42H.

Now that the module is programmed as a PWM, a value must be loaded in the high byte of the compare register to select the duty cycle. The value can be any number from 0 to 255. In the 83C51FA loading 0 in the CCAPnH will yield 100% duty cycle, and 255 (0FFh) will generate a 0.4% duty cycle. See figure 5.

The next step is to start the PCA timer. The bit that turns the timer on and off is called CR and is bit 6 of

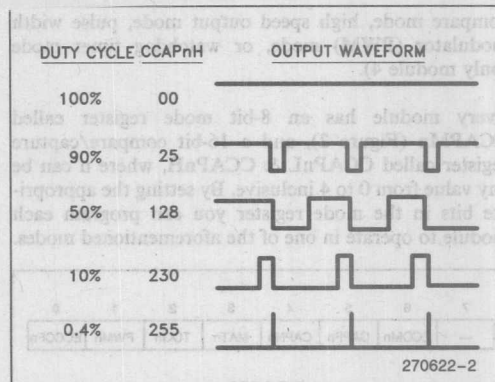


Figure 5. Selected Duty Cycles and Waveforms

CCON register (Figure 3). Since this register is bit addressable, you can use bit instructions to turn the timer on and off.

In the following example module 2 has been selected to provide a PWM signal to a motor driver. An external clock will be provided for the timer input, so the value that needs to be loaded into CMOD is 06H.

HARDWARE REQUIREMENT

When using an 83C51FA, very little hardware is required to control a motor. The controller can interface to the motor through a driver as shown in figure 6.

MOV	CMOD,#06	; timer input external
MOV	CCAPM2,#42H	; put the module in PWM mode.
MOV	CCAP2H,#0	; 0 provides 100% duty cycle (5V)
SETB	CR	; turn timer on
...		
...		
...		
END		

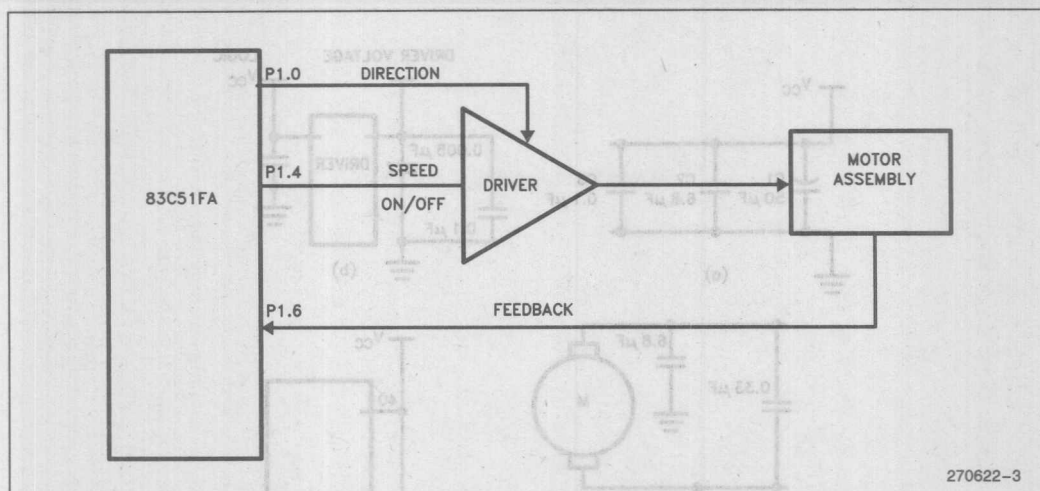


Figure 6. Simplified Circuit Diagram of a Closed Loop System

This configuration, a closed loop circuit, takes up only three I/O pins. The line controlling direction can be a regular port pin but the speed control line must be one of the port 1 pins which corresponds to a PCA module selected for PWM. Depending on how the feedback is generated and processed, it could be connected to a regular I/O, an external interrupt, or a PCA module. Feedback is discussed in more detail in the feedback section of this application note.

The diagram in Appendix A is an example of a DC motor circuit which has been built and bench-tested.

DRIVER CIRCUIT

Although some DC motors operate at 5 volts or less, the 83C51FA can not supply the necessary current to drive a motor directly. The minimum current requirements of any practical motor is higher than any microcontroller can supply. Depending on the size and ratings of the motor, a suitable driver must be selected to take the control signal from the 83C51FA and deliver the necessary voltage and current to the motor.

A motor draws its maximum current when it is fully loaded and starts from a stand still condition. This factor must be taken into account when choosing a driver. However, if the application requires reversing the motor, the current demand will even be higher. As the motor's speed increases, its power consumption decreases. Once the speed of a motor reaches a steady state, the current depends on the load and the voltage across the motor.

Standard motor drivers are available in many current and voltage ratings. One example is the L293 series which can output up to 1 ampere per channel with a supply voltage of 36 V. It has separate logic supply and takes logical input (0 or 1) to enable or disable each channel. There are four channels per device. The L293D also includes clamping diodes needed for protecting the driver against the back EMF generated during the reversing of motor.

NOISE CONSIDERATIONS

Motors generate enough electrical noise to upset the performance of the controller. The source of the noise could be from the switching of the driver circuits or the motor itself. Whatever the cause of the noise may be, it must be isolated or bypassed.

Isolating the microcontroller from the driver circuit is helpful in keeping the noise limited.

Bypass capacitors help a great deal in suppressing the noise. They must be added to the power and ground (Figure 7 diagram a), on the driver circuit (diagram b), on the motor terminals (diagram c), and on the 83C51FA (diagram d). The capacitors must be as close to the component as possible. In fact the best location is under the chip or on top of it if packaging allows. The diagrams in figure 7 show the location and some typical values for the bypass capacitors.

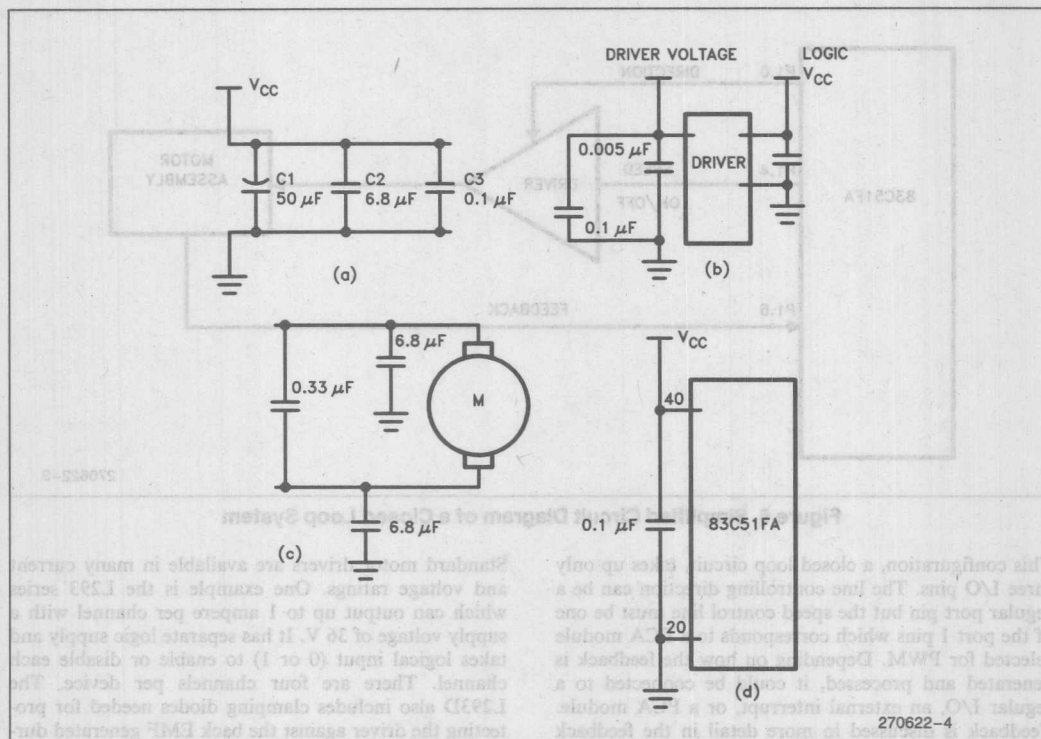


Figure 7. Typical Locations and Values for Bypass Capacitors

OPEN LOOP & CLOSED LOOP SYSTEMS

There are two types of motor control systems: open loop and closed loop.

In the open loop system the controller outputs a signal to turn the motor on/off or to change the direction of the rotation based on an input that does not come from the motor. For example, the position of a manual or timer switch becomes the input to the controller, which varies the input to the motor. In another case, the controller may take input from data tables in the program to run, vary the speed, reverse direction, or stop the motor.

Closed loop systems can use one or more of the above mentioned examples for the open loop system, plus at least one feedback signal from the motor. The feedback signal provides such information as speed, position, and/or direction of motion.

Many applications require that a motor run at a constant speed. The controller has to continuously make adjustments to keep the speed within the limits. In some cases the speed of the motor is synchronized to another motor or moving part of the system.

Depending on the type of feedback signal, the 83C51FA may have to use other modules of the PCA along with other on-chip peripherals such as Timer/Counters, Serial Port, and the interrupt system to precisely control a DC motor.

The example in the following section uses one PCA module to generate PWM, and another module (in capture mode) to receive feedback from a DC motor.

FEEDBACK

The feedback comes from a sensing device which can detect motion. The sensing device may be an optical encoder, infrared detector, Hall effect sensor, etc. Depending on the application, one or more of the above mentioned sensing devices may be suitable.

The optical sensors should be encapsulated for better reliability. If they are not enclosed, factors such as ambient light, dust, and dirt can lessen their sensitivity.

Hall effect sensors are insensitive to any type of light. They change logic levels going into and coming out of a magnetic field. The sensing device is normally mounted

on some stationary part of the system and the magnet is installed on the rotating part. The potential problem with the Hall effect sensors are that if the gap between the magnet and the sensing device is too big, the sensing device may not be affected by the magnetic field. Also the number of magnets is limited which means fewer feedback pulses will be provided.

Whatever the means of sensing, the result is a signal which is fed to the controller. The 83C51FA can use the feedback signal to determine the speed and position of the motor. Then it can make adjustments to increase or decrease the speed, reverse the direction, or stop the motor.

In the following example module 3 of PCA is set up to perform in the capture mode. In this mode module 3 will receive feedback signals from a Hall effect transistor fixed behind a wheel which is mounted on the shaft of a DC motor. Two magnets are embedded on this wheel in equal distances from each other (180 degrees apart). Every time that the Hall effect transistor passes through the magnetic field, it generates a pulse.

The signal is input to P1.6 which is the external interface for module 3 of the PCA. In this example, module 3 is programmed to capture on the rising edge of the

input signal. The time between the two captures corresponds to $\frac{1}{2}$ of a revolution. Thus, two consecutive captures can provide enough information to calculate the speed of the motor as explained in the next paragraph. By storing the value of the capture registers each time, and comparing it to its previous value, the controller can constantly measure and adjust the speed of the motor. Using this method one can run a motor at a precise speed, or synchronize it to another event.

In the PCA interrupt service routine, each capture value is stored in temporary locations to be used in a subtract operation. Subtracting the first capture from the second one will yield a 16-bit result. The resultant value, which will be referred to as "Result" in the rest of this document, is in PCA timer counts. An actual RPM can be calculated from Result. Although the 83C51FA can do the calculation, it would be much faster to provide a lookup table within the code. The table will contain values which have been calculated for a possible range of Results.

The following code is an example of how to measure the period of a signal input to module 3 of the 83C51FA. The diagram in figure 8 shows how the period corresponds to the rotation of the wheel. In the diagram "T" is the period and "t" is the time that the magnet is passing in front of the Hall effect transistor.

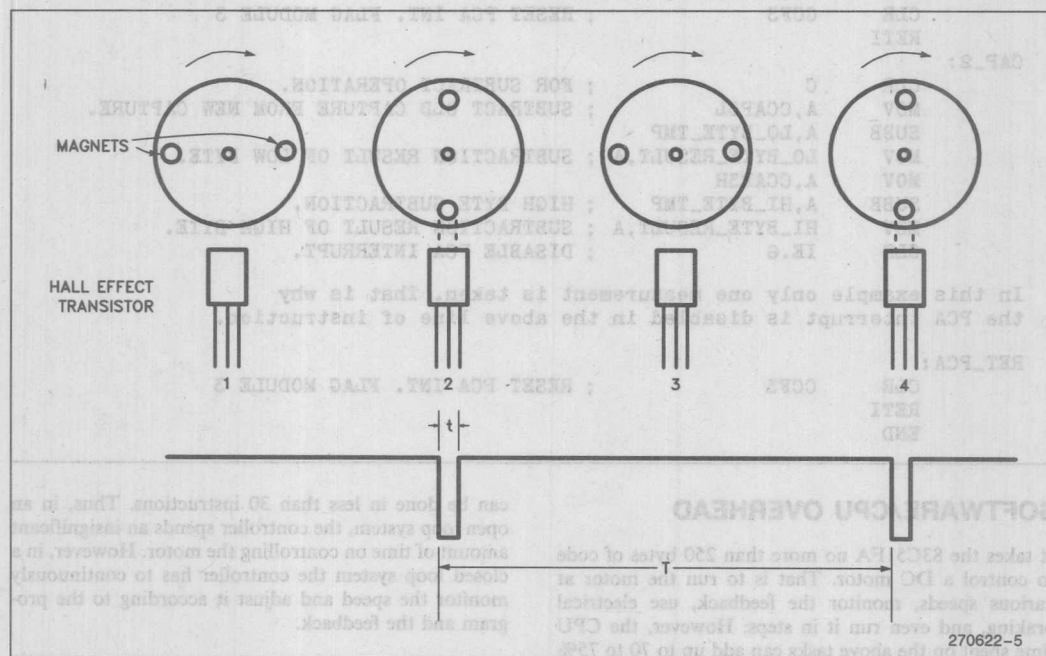


Figure 8. The Output Waveform of the Hall Effect Transistor as it goes Through the Magnetic Field

```

FLAG BIT 0 ; test flag
HI_BYTE_TMP DATA 45H
LO_BYTE_TMP DATA 46H
HI_BYTE_RESULT DATA 47H
LO_BYTE_RESULT DATA 48H

ORG OOH
JMP BEGIN

ORG 33H
JMP PCA_ISR

BEGIN:
MOV CMOD,#0 ; SET PCA TIMER INPUT fOSC/12.
MOV CCAPM3,#21H ; MODULE 3 IN POSITIVE CAPTURE MODE.
MOV CCAP3H,#9AH ; PWM AT 60 PERCENT DUTY CYCLE.
SETB IP.6 ; SET PCA INT. AT HIGH PRIORITY.
MOV IE,#0COH ; ENABLE PCA INTERRUPT.
CLR FLAG
SETB CR ; TURN PCA TIMER ON.
.
.
.
PCA_ISR:
JB FLAG,CAP_2 ; FLAG BIT IS SET TO SIGNIFY 1st
SETB FLAG ; CAPTURE COMPLETE.
MOV HI_BYTE_TMP,CCAP3H; SAVE FOR NEXT CALCULATION.
MOV LO_BYTE_TMP,CCAP3L
CLR CCF3 ; RESET PCA INT. FLAG MODULE 3
RETI

CAP_2:
CLR C ; FOR SUBTRACT OPERATION.
MOV A,CCAP3L ; SUBTRACT OLD CAPTURE FROM NEW CAPTURE.
SUBB A,LO_BYTE_TMP
MOV LO_BYTE_RESULT,A ; SUBTRACTION RESULT OF LOW BYTE.
MOV A,CCAP3H
SUBB A,HI_BYTE_TMP ; HIGH BYTE SUBTRACTION.
MOV HI_BYTE_RESULT,A ; SUBTRACTION RESULT OF HIGH BYTE.
CLR IE.6 ; DISABLE PCA INTERRUPT.

In this example only one measurement is taken. That is why
the PCA interrupt is disabled in the above line of instruction.

RET_PCA:
CLR CCF3 ; RESET PCA INT. FLAG MODULE 3
RETI
END

```

SOFTWARE/CPU OVERHEAD

It takes the 83C51FA no more than 250 bytes of code to control a DC motor. That is to run the motor at various speeds, monitor the feedback, use electrical braking, and even run it in steps. However, the CPU time spent on the above tasks can add up to 70 to 75% of the total time available (clock frequency 12 MHz).

The section of software which turns the motor on and off, or sets the speed is very short. In fact, all of that

can be done in less than 30 instructions. Thus, in an open loop system, the controller spends an insignificant amount of time on controlling the motor. However, in a closed loop system the controller has to continuously monitor the speed and adjust it according to the program and the feedback.

The rest of this section talks about electrical braking, stepping a DC motor, and offers examples of code to implement these techniques.

ELECTRICAL BRAKING

Once a DC motor is running, it picks up momentum. Turning off the voltage to the motor does not make it stop immediately because the momentum will keep it turning. After the voltage is shut off, the momentum will gradually wear off due to friction. If the application does not require an abrupt stop, then by removing the driving voltage, the motor can be brought to a gradual stop.

An abrupt stop may be essential to an application where the motor must run a few turns and stop very quickly at a predetermined point. This could be achieved by electrical braking.

Electrical braking is done by reversing the direction of the motor. In order to run in reverse direction, the motor has to stop first, at which time the driving voltage is eliminated so that the motor does not start in the new direction. Therefore the length of time that the reversing voltage is applied must be precisely calculated to ensure a quick stop while not starting it in the reverse direction.

There is no simple formula to calculate when to start, and how long to maintain braking. It varies from motor to motor and application to application. But it can be perfected through trial and error.

In a closed loop system, the feedback can be used to determine where or when to start braking and when to discontinue.

During the electrical braking, or any time that the motor is being reversed, it draws its maximum current. To a motor which is turning at any speed, reversing is a heavy load. The current demand of a motor, when it has been reversed, is much higher than when it has just been powered on.

The following shows a code sample for electrical braking on a DC motor. The code is designed for the hardware shown in Appendix A. The subroutine DELAY provides the period that the reverse voltage is applied to the motor. The code for this subroutine is available in the TIME DELAYS section of this document.

2

BEGIN:

```
MOV    CMOD,#0      ; SET PCA TIMER INPUT fOSC/12.
MOV    CCAPM1,#42H   ; SETTING THE MODULE TO PWM MODE.
SETB   CR            ; PCA TIMER RUN.
```

; DRIVE MOTOR CLOCKWISE

```
CLR    P1.0          ; P1.0 AND THE PWM OF MODULE 1-
MOV    CCAP1H,#00     ; CONTROL THE SPEED AND DIRECTION.
```

; 00 IN THIS REGISTER PUTS OUT MAX PWM (LOGICAL 1)

```
CALL   STOP_MOTOR
```

.

.

STOP_MOTOR:

```
SETB   P1.0          ; REVERSING THE MOTOR.
MOV    CCAP1H,#0FFH   ;
CALL   DELAY          ; WAITING FOR 0.5 SECOND.
CLR    P1.0          ; REDUCING VOLTAGE TO 0.
RET     ; RETURN FROM SUBROUTINE.
```

.

.

STEPPING A DC MOTOR

Using the 83C51FA, it is possible to run a simple DC motor in small steps. The resolution of the steps will be as high as the resolution of the encoder. If this resolution is sufficient, here is a technique to run a DC motor in steps.

Using a gear box to gear down the motor will increase the resolution of steps. However, putting too much load through the gears will cause sluggish starts and stops.

Electrical braking is used in order to stop the motor at each step. Therefore, the routine that runs the motor in steps will consist of turning it on with full force, waiting for certain period, and stopping it as fast as possible. The wait period depends on the number of steps per revolution.

As the steps and the intervals between them become smaller, the average current demand of the motor increases. This is because the motor is operated at its maximum torque condition every time it starts to rotate and every time it is reversed for electrical braking.

The following code sample shows a continuous loop which runs the motor in steps. The number of steps per revolution depends on the duration of the delay generated by DELAY subroutine. Subroutine WAIT provides the time between the steps.

Subroutine DELAY is the period of time that the motor is kept in reverse. This period must be determined through trial and error for each type of motor and system.

TIME DELAYS

While the 83C51FA is controlling a motor it must frequently wait for the motor to move to certain position before it can proceed with the next task. For example, in the case of electrical braking when the controller reverses the polarity of voltage across the motor, depending on the type, size, and the speed of the motor, it may have up to a second of CPU time before it will turn the motor off.

The wait may be implemented in different ways. Any of the Timer/Counters or unused PCA modules could be utilized to provide accurate timing. The advantage in using the timers is that while the timer is counting, the processor can be taking care of some other tasks. When the timer times out and generates an interrupt the processor will go back and continue servicing the motor.

If there are no timers or PCA modules available for this purpose, a software timer maybe set up by decrementing some of the internal registers. In this method the processor will be tied up counting up or down and will not be able to do anything else. An example of such a timer is:

LOOP:

```
CLR    P1.0          ; SET DIRECTION CLOCKWISE
MOV    CCAP1H,#0      ; MAX PWM
```

The above instruction sets the motor running clockwise. The controller can be doing other tasks if need be, or just stay in a wait loop, then stop the motor as shown below.

```
SETB   P1.0          ; REVERSING THE MOTOR.
MOV     CCAP1H,#OFFH  ;
CALL    DELAY         ; WAIT FOR IT TO STOP.
CLR     P1.0          ; REDUCE VOLTAGE TO 0.
CALL    WAIT          ; TIME BEFORE NEXT STEP.
JMP     LOOP
```

```

DELAY:
      MOV     R4,#25      ;(decimal)
      MOV     R5,#255     ;(decimal)
DELAY_LOOP:
      DJNZ    R5,DELAY_LOOP
      DJNZ    R4,DELAY_LOOP
      RET

```

Subroutine DELAY provides approximately 6.4 ms with a 12 MHz clock or 4.8 milliseconds with a 16 MHz clock. The length of this delay can be controlled by loading smaller or larger values to R4 to vary from 260 microseconds up to 65 milliseconds at 12 MHz or 48 milliseconds at 16 MHz oscillator frequency. Larger delays may be obtained by cascading another register and creating an outer loop to this one.

Let us assume that it will take a motor 500 milliseconds to stop from its CW rotation and we are going to use Timer/Counter 0 to provide the wait period. Subroutine DELAY1 will keep track of this timing. Module 1 of PCA is selected to provide the PWM.

2

```

ORG     OBH
JMP     TIMER_INTERRUPT_ROUTINE
      .
      CLR     P1.0      ; SET DIRECTION CW
      MOV     CCAP1H,#0  ; MAX PWM

```

Now the motor is running and the controller can do other tasks.
Some typical tasks are called in the following segment.

```

BUSY_LOOP:
      CALL    MONITOR_DISPLAY
      CALL    SCAN_KEY_BOARD
      CALL    SCAN_INPUT_LINES
      JNB     STOP_FLAG,BUSY_LOOP

```

STOP_FLAG gets set by a feedback signal and denotes that the motor must stop.

```

      SETB    P1.0      ; REVERSING THE MOTOR
      MOV     CCAP1H,#OFFH
      CALL    DELAY      ; WAIT TILL MOTOR STOPS
      CLR     P1.0      ; REDUCE VOLTAGE TO 0
      .
      .
      .

```

```

DELAY1:
      SETB    EA
      SETB    ETO        ; enable timer 0 interrupt
      MOV     TLO,#0D8H
      MOV     TH0,#5EH

```

```

SETB     TRO      ; timer 0 on
MOV      R7,#8    ; keep track of how many times
; timer 0 must roll over

; continue performing other tasks

```

MONITOR_LOOP:

```

CALL     MONITOR_DISPLAY
CALL     SCAN_KEY_BOARD
CALL     SCAN_INPUT_LINES
JB       TRO,MONITOR_LOOP
RET

```

TIMER_INTERRUPT_ROUTINE:

```

DJNZ     R7,FULL_COUNT
CLR      TRO

```

FULL_COUNT

```

RETI

```

To implement a 500 milliseconds delay, timer 0 is used here. In mode 1 timer 0 is a 16-bit timer which takes 65.535 milliseconds at 12 MHz to roll over. Dividing 500 milliseconds to 65.535 shows that timer has to overflow more than 7 times but less than 8 times. How much more than 7 times? The following calculation yields the initial load value of the timer.

$$500 \div 65.535 = 7.2695 \text{ taking it backward}$$

$$65.535 \times 7 = 458.745 \text{ milliseconds}$$

$$500 - 458.745 = 41.255 \text{ milliseconds or } 41255 \text{ microseconds.}$$

In hexadecimal it is A127H. The initial load value is the complement of this value which is 5ED8H.

CONCLUSION

The 83C51FA with all its on-chip peripherals is a system on one chip. It can simplify the design of a control board and reduce the chip count. Up to 5 DC motors can be controlled while doing other tasks such as monitoring feedback lines, human interfacing (scanning a keyboard, displaying information), and communicating with other processors. The MCS-51 powerful instruction set provides maximum flexibility with minimum hardware.

With its onboard program memory capability, the need for the external EPROM and address latch is eliminated. The 83C51FA can have up to 8K bytes of code and the 83C51FB can have up to 16K bytes of code on-board.

This microcontroller can be used in industrial, commercial, and automotive applications.

APPENDIX A

Figure A-1 shows a symbolic view of the L293B driver. This driver has 4 channels but only two are shown here. Note the inputs A and B and how they are related to each other. You can input the PWM to either one of the inputs and by toggling the other input start or stop the motor. While running, the PWM input controls the

speed. Pin P1.4 corresponds to module 1 of the PCA, and pin P1.0 is used as a regular I/O pin.

Figure A-2 shows the schematic of the motor driver, motor, feedback path, and the supporting components.

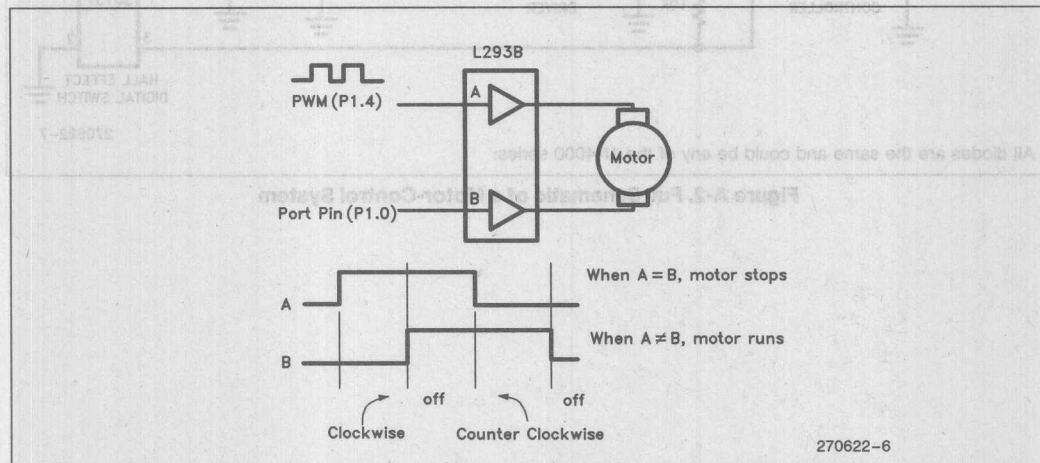


Figure A-1. The L293B Motor Driver

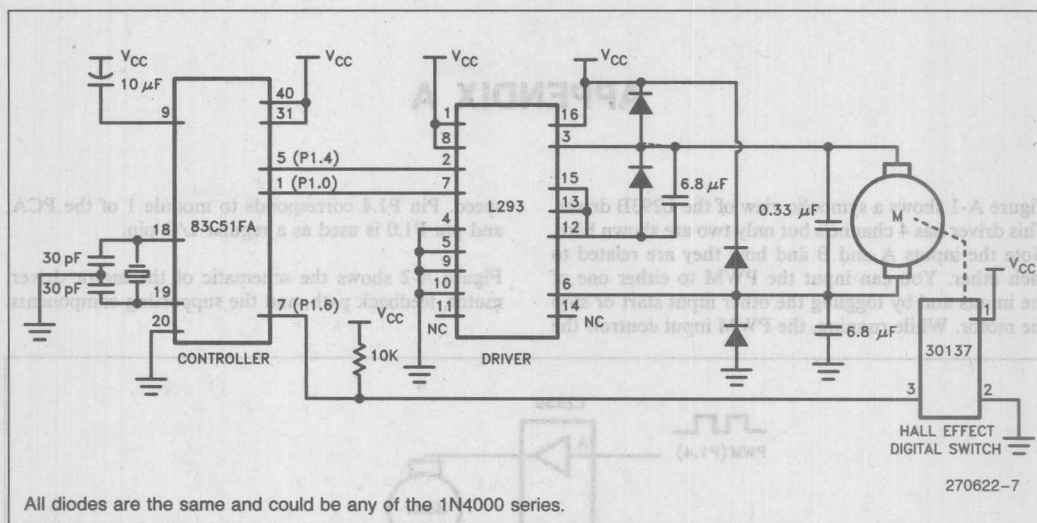


Figure A-2. Full Schematic of a Motor-Control System

Using the 8051 Microcontroller with Resonant Transducers

TOM WILLIAMSON

Abstract—Having to interface an analog transducer to a digital control system through an analog-to-digital converter represents an expensive bottleneck in the development of many systems. Some transducer companies are addressing this problem by developing proprietary families of resonant transducers.

Resonant transducers are oscillators whose frequency depends in some known way on the physical property being measured. The electrical output from these devices is a train of rectangular pulses whose repetition rate encodes the value of the measurand. Changes in the measurand cause the frequency to shift. The microcontroller detects the frequency shift, runs a validity check on it, and converts it in software to the measurand value.

This paper discusses software interfacing techniques between resonant transducers and the 8051. Techniques for measuring frequency and period are discussed and compared for resolution and interrogation time. The 8051 is capable of performing these tasks in extremely short CPU time. Requirements for obtaining n -bit resolution in the measurement are discussed. It is determined that it is always faster to evaluate the measurand to a given level of resolution by measuring the period rather than the frequency, even if the measurand is proportional to the frequency rather than to the period. Numerical and software examples are presented to illustrate the concepts.

I. RESONANT TRANSDUCERS

MOST sensing transducers are not directly compatible with digital controllers, because they generate analog signals. A few transducer companies are developing proprietary families of sensors which generate signals that are more directly compatible with digital systems. These are not analog sensors with built-in A-D conversion, but oscillators whose frequency depends in some known way on the physical property being measured.

The technology is applicable to virtually any type of measurand: pressure, gas density, position, temperature, force, etc. The sensor and microcontroller can operate from the same supply voltage, so the sensor can in most cases connect directly to a port pin on the microcontroller.

The nominal reference frequency of the output signal from these devices is in the range of 20 Hz–500 kHz, depending on the design. A change in the measurand away from the reference condition causes the frequency to shift by an amount that is related to the change in the measurand value. Transducers are available that have a full-scale frequency shift of 2–1. The microcontroller detects the change in frequency or period and converts it in software to the measurand value.

II. CONNECTING THE DIGITAL TRANSDUCER TO THE 8051

Normally the transducer output can be connected directly to one of the 8051 port pins. An exception would occur when the

Manuscript received October 25, 1984.

The author is with Intel Corporation, Chandler, AZ 85224.

transducer signal does not restrict itself to the voltage range of -0.5 to $+5.5$ V.

The 8051 is not sensitive to the rise and fall times of its input signals. It detects transitions by sampling its port pins at fixed intervals (once per machine cycle), and responds to a change in the sequence of samples. If the slew rate of the transducer signal is extremely slow, noise superimposed on the signal could cause the sequence of samples to show false transitions. There could on that account be situations in which the transducer signal should be buffered through a Schmitt Trigger to square it up.

III. TIMER/COUNTER STRUCTURE IN THE 8051

The 8051 has two 16-bit timer/counters: Timer 0 and Timer 1. Both can be configured in software to operate either as timers or as event counters.

In the "timer" function, the register is automatically incremented every machine cycle. Since a machine cycle in the 8051 consists of 12 clock periods, the timer is being incremented at a constant rate of $1/12$ the clock frequency.

In the "counter" function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin (T0 or T1). The way this function works is the external input pin is sampled once each machine cycle (once every 12 clock periods), and when the samples show a high in one cycle and a low in the next, the count is incremented.

Note too that since it takes two machine cycles (24 clock periods) to recognize a 1-to-0 transition, the maximum count rate is $1/24$ the clock frequency. If the clock frequency is 12 MHz, the maximum count rate is 500 kHz. There are no requirements on the duty cycle of the signal being counted.

The 8052, an enhanced version of the 8051, has three 16-bit timer/counters, two of which are identical to those in the 8051. The third timer/counter can operate either as a 16-bit timer/counter with automatic reload to a preset 16-bit value on rollover, or as a 16-bit timer/counter with a "capture" mode. In the capture mode a 1-to-0 transition at the T2EX pin causes the current value in the counting register to the "captured" into RAM. The third timer makes the 8052 particularly easy to interface with resonant transducers.

IV. WHETHER TO MEASURE FREQUENCY OR PERIOD

Measuring the frequency requires counting transducer pulses for a fixed sample time. Measuring the period requires measuring elapsed time for a fixed number of transducer pulses. For a given level of accuracy in the determination of the value of the measurand, it is usually faster to measure the period, rather than the frequency, even if the measurand is

proportional to frequency rather than period. However, both types of measurements will be discussed here.

Two timer/counters can be used, one to mark time and the other to count transducer pulses. If the frequency being counted does not exceed 50 kHz or so, one may equally well connect the transducer signal to an external interrupt pin, and count transducer pulses in software. That frees one timer, with very little cost in CPU time.

V. HOW TO MEASURE TRANSDUCER FREQUENCY

Measuring the frequency means counting transducer pulses for some desired sample time. The count that is directly obtained is $T \times F$, where T is the sample time and F is the frequency. The full scale range is $T \times (F_{\max} - F_{\min})$. For n -bit resolution

$$1 \text{ LSB} = \frac{T \times (F_{\max} - F_{\min})}{2^n}$$

Therefore, the sample time required for n -bit resolution is

$$T = \frac{2^n}{F_{\max} - F_{\min}}$$

For example, 8-bit resolution in the measurement of a frequency that varies between 5 and 10 kHz would require, according to this formula, a sample time of 51.2 ms. The maximum acceptable frequency count would be $51.2 \text{ ms} \times 10 \text{ kHz} = 512$ counts. The minimum would be 256 counts. Subtracting 256 from each frequency count would allow the frequency to be reported on a scale of 0 to FF in hex digits.

If F_{\min} and F_{\max} are closer together it takes more time to resolve them. 8-bit resolution in the measurement of a frequency that varies between 7 and 9 kHz would require a sample time of 128 ms. The maximum and minimum acceptable counts would be 1152 and 896. Subtracting 896 from each frequency count would allow the frequency to be reported on a scale of 0 to FF in hex digits.

To implement the measurement, one timer is used to establish the sample time. In this function it autoincrements every machine cycle. A machine cycle consists of 12 periods of the clock oscillator. The sample time can be converted to machine cycles by multiplying it by $(F_{\text{xtal}})/12$, where F_{xtal} is the 8051 clock frequency. The timer needs to be preset so that it rolls over at the end of each sample time. Then it generates an interrupt, and the interrupt routine reads and clears the transducer pulse counter, and then reloads the timer with the correct preset value.

The preset or reload value is the two's complement negative of the sample time in machine cycles. For example, with a 12-MHz clock frequency, the reload value required to establish a 51.2 ms sample time is

$$\frac{(51.2 \text{ ms}) \times (12000 \text{ kHz})}{12} = -51200 = 3800 \text{ H.}$$

In many cases the required sample time exceeds the capacity of a 16-bit timer. For example, establishing a 128 ms sample time with a 12-MHz clock frequency requires a 3-byte timer with a reload of FE0C00H. The 8051 timer, being only 2-

bytes wide, can be augmented in software in the timer interrupt routine to three bytes. The 8051 has a DJNZ instruction (decrement and jump if not zero) which makes it easier to code the third timer byte to count down instead of up. If the third timer byte counts down, its reload value is the two's complement of what it would be for an up-counter. For example, if the two's complement of the sample time is FE0C00H, then the reload value for the third timer byte would be 02, instead of FE. The timer interrupt routine might then be

```
DJNZ THIRD_TIMER_BYTE,OUT
MOV TLO,#0
MOV TH0,#0CH
MOV THIRD_TIMER_BYTE,#02
```

(Now read and clear the transducer pulse counter.)

OUT: RETI

Interrupt latency will have no effect on the measurement if the latency is the same for every sample time.

The trouble with measuring the frequency is it is not only slow, but a waste of the resolving power of the 8051's timers. A timer with microsecond resolution is being used to mark off 100-ms time periods. The technique is nevertheless useful if the timer is already serving other purposes (servicing a display, perhaps), so that the sample time is coming relatively free of charge. But in most cases it is faster and equally accurate to measure the frequency by deriving it from a measurement of the period.

VI. HOW TO MEASURE THE PERIOD

Measuring the period of the transducer signal means measuring the total elapsed time over N -transducer pulses. The quantity that is directly measured is $N \times T$, where T is the period of the transducer signal in machine cycles. The relationship between T in machine cycles and the transducer frequency F in arbitrary frequency units is

$$T = \frac{F_{\text{xtal}}}{F} \times (1/12)$$

where F_{xtal} is the 8051 clock frequency, in the same unit as F .

The full scale range then is $N \times (T_{\max} - T_{\min})$. For n -bit resolution

$$1 \text{ LSB} = \frac{N \times (T_{\max} - T_{\min})}{2^n}$$

Therefore, the number of periods over which the elapsed time should be measured is

$$N = \frac{2^n}{T_{\max} - T_{\min}}$$

However, N must also be an integer. It is logical to evaluate the above formula (do not forget that T_{\max} and T_{\min} have to be in machine cycles) and select for N the next higher integer. This selection gives a period measurement that has somewhat more than n -bit resolution, which may or may not be acceptable, depending on the overall requirements of the

system. It can be scaled back to n -bit resolution, if necessary, by the following computation:

$$\text{reported value} = \frac{NT - NT_{\min}}{NT_{\max} - NT_{\min}}$$

where NT is the elapsed time measured over N periods.

The computation can be done in math if a suitable divide routine is available in the software. For 8-bit resolution it is entirely reasonable to find the reported value in a look-up table, which would take up somewhat more than one page in ROM. In fact, the look-up table would contain $NT_{\max} - NT_{\min}$ entries.

For example, suppose we want 8-bit resolution in the measurement of the period of a signal whose frequency varies from 5 to 10 kHz. If the clock frequency is 12 MHz, then T_{\max} is $(12\,000\text{ kHz})/(12 \times 5\text{ kHz}) = 200$ machine cycles, and T_{\min} is 100 machine cycles. The timer needs to be on then for $N = 2.56$ periods, according to the formula. Using $N = 3$ periods will give maximum and minimum NT values of 600 and 300 machine cycles. This is somewhat more than 8-bit resolution. It can be scaled to 8 bits with a 300-byte look-up table, if desired.

To implement the measurement, one timer is used to measure the elapsed time NT . Enabling its interrupt is optional. The timer interrupt could be used to indicate a short or open in the transducer circuit.

Then the transducer is connected to one of the external interrupt pins (INT0 or INT1), and this interrupt is configured to the transition-activated mode. In the transition-activated mode every 1-to-0 transition in the transducer output will generate an interrupt. The interrupt routine counts transducer pulses, and when it gets to the predetermined N , it reads and clears the timer. For example

```
DJNZ PULSES,OUT
MOV PULSES,N,PERIODS
(Read and clear timer.)
OUT: RETI
```

If other interrupts are also to be enabled, the one connected to the transducer should be set to Priority 1, and the others to Priority 0. This is to control the interrupt response time. The response time will not affect the measurement if it is the same for every measurement. Variations in the response time will, however, affect the measurement. Setting the pulse-counter interrupt to Priority 1 and all others to Priority 0 will minimize variations in the response time. The response time will then be limited to range from 3 to 8 machine cycles.

VII. PULSEWIDTH MEASUREMENTS

The 8051 timers have an operating mode which is particularly suited to pulsewidth measurements, and may be useful here if the transducer has a fixed duty cycle, or if the transducer output is pulsewidth modulated instead of frequency modulated by the measurand.

In this mode the timer is turned on by the on-chip circuitry in response to an input high at the external interrupt pin, and off by an input low. The external interrupt itself is enabled, so the same 1-to-0 transition from the transducer that turns off the

timer also generates an interrupt. The interrupt routine would then read and reset the timer.

The advantage of this method is that the transducer signal has direct access to the timer gate, with the result that variations in the interrupt response time cease to be a factor. The timer can be read and cleared any time before the next high in the transducer output.

VIII. DERIVING FREQUENCY FROM A PERIOD MEASUREMENT

We now consider the problem of measuring the transducer frequency to n -bit resolution by deriving it from a direct measurement of the period. The advantage of this technique is speed. It is always faster to measure period than frequency. But it is important to end up with a frequency value that has the same resolution and accuracy as a directly measured frequency. Two questions need to be addressed.

- 1) To achieve n -bit resolution in the calculated frequency, how much resolution is required in the period?
- 2) Having measured the period to the required resolution, what is the most efficient way to calculate the frequency?

These questions will be addressed one at a time.

IX. RESOLUTION REQUIREMENTS

In general, n -bit resolution in the frequency derivation requires somewhat more than n -bit resolution in the period measurement. How much more? It will be demonstrated presently that if the transducer frequency varies over a 2-to-1 range, the frequency can be calculated with n -bit resolution from period measurements that have $(n + 1)$ -bit resolution.

The more practical form of the question is over how many periods (N) must the transducer signal be sampled to obtain the required resolution in F ? And so, we commence a calculation of N .

The basic calculation of frequency from $N \times T$ (which we shall call NT) is straightforward:

$$F = N/(NT).$$

The relationship between an increment dF in the calculated frequency due to an increment $d(NT)$ in the measured period is, therefore,

$$\begin{aligned} dF &= -\frac{N}{(NT)^2} d(NT) \\ &= -\frac{F^2}{N} d(NT). \end{aligned}$$

This equation says the value of an LSB in the calculated frequency is $(F^2)/N \times$ the value of an LSB in NT . Then the maximum value that an LSB in the calculated frequency can have is $(F_{\max})^2/N \times$ the value of an LSB in NT . For the calculated frequency to have n -bit resolution over the entire range of frequencies, the value of its LSB must never exceed $(F_{\max} - F_{\min})/2^n$. Therefore, the measurement requires

$$\frac{(F_{\max})^2}{N} \times (1 \text{ LSB in } NT) \leq \frac{F_{\max} - F_{\min}}{2^n}.$$

WILLIAMSON: USING THE 8051 MICROCONTROLLER

The required resolution in NT is, therefore,

$$1 \text{ LSB in } NT \leq \frac{N \times (F_{\max} - F_{\min})}{2^n \times (F_{\max})^2}$$

Now, to say that NT is measured with m -bit resolution means

$$1 \text{ LSB in } NT = \frac{N \times (1/F_{\min} - 1/F_{\max})}{2^m}$$

Substituting this value for 1 LSB into the required resolution and solving for 2^m yields

$$2^m \geq \frac{F_{\max}}{F_{\min}} \times 2^n$$

Then the requirement on m is

$$m \geq n + \frac{\ln(F_{\max}/F_{\min})}{\ln(2)}$$

It can be stated with some certainty, then, that if the transducer frequency varies over a range of 2-to-1, the frequency can be calculated with 8-bit resolution from a period measurement that has 9-bit resolution. If the frequency variation is less than 2-to-1, another full bit of resolution in the period measurement is not needed.

To obtain m -bit resolution in NT , N must satisfy

$$N \geq \frac{2^m}{T_{\max} - T_{\min}}$$

Substituting for 2^m , and using $T_{\max} = 1/F_{\min}$ and $T_{\min} = 1/F_{\max}$, gives the result that

$$N \geq \frac{(F_{\max})^2}{F_{\max} - F_{\min}} \times 2^n$$

It should be noted that the units of frequency here are periods/machine cycle, since the 8051 measures time by counting machine cycles. The conversion factor between Hz and periods/machine cycle is $12/(\text{clock frequency})$. So the requirement on N can also be written

$$N \geq \frac{F_{\max}}{F_{\max} - F_{\min}} \times \frac{F_{\max}}{F_{\text{xtal}}} \times 12 \times 2^n$$

where F_{xtal} is the 8051 clock frequency in the same units as F_{\max} and F_{\min} . This is the number of transducer pulses over which the transducer signal must be sampled to achieve the required resolution in F .

For example, suppose that 8-bit resolution is required in F , where $F_{\max} = 10 \text{ kHz}$ and $F_{\min} = 5 \text{ kHz}$, and that $F_{\text{xtal}} = 12 \text{ MHz}$. Then the above calculation shows that $N = 6$ periods gives sufficient resolution in the period measurement to satisfy the resolution requirement in F . Six periods will take 0.6–1.2 ms of sampling time, on that frequency range. Recall that the sample time for a direct frequency measurement of the same signal and to the same resolution was earlier calculated to be 51.2 ms.

X. COMPUTING THE FREQUENCY FROM THE PERIOD

The period measurement leaves one with a 16-bit integer, which is $N \times T$ (or NT) in machine cycles. The conversion to frequency is straightforward:

$$F = N/(NT) \text{ periods/machine cycle.}$$

The quantity of interest is probably not F , but a normalized measure of the amount by which F exceeds its minimum acceptable value. This quantity represents, through the transducer's transfer function, the "reported value" of the measurand, and this quantity is an n -bit integer whose value ranges from 0 (all bits = 0) to full scale (all bits = 1). This normalized frequency is

$$f = \frac{F - F_{\min}}{F_{\max} - F_{\min}} = \frac{F_{\min}}{F_{\max} - F_{\min}} \times (F/F_{\min} - 1)$$

Using $F = N/(NT)$ and $F_{\min} = N/(NT_{\max})$ allows the normalized frequency to be written

$$f = \frac{F_{\min}}{F_{\max} - F_{\min}} \times \frac{NT_{\max} - NT}{NT}$$

To get a handle on what kinds of numbers are involved here, consider the situation where 8-bit resolution is required in f , and in which $F_{\text{xtal}} = 12 \text{ MHz}$, $F_{\max} = 10 \text{ kHz}$, and $F_{\min} = 5 \text{ kHz}$. We have previously determined that for this set of conditions, $N = 6$ periods gives sufficient resolution in the period measurement to satisfy the resolution requirement in F (and in f). With a 12 MHz clock frequency, T_{\max} in machine cycles is $(12\,000 \text{ kHz})/(12 \times 5 \text{ kHz}) = 200$ machine cycles, so NT_{\max} is $6 \times 200 = 1200$ machine cycles. The calculation for f then becomes

$$f = \frac{1200 - NT}{NT}$$

The minimum acceptable value that NT can have is $(N \times T_{\min} + 1)$, where $T_{\min} = (12\,000 \text{ kHz})/(12 \times 10 \text{ kHz}) = 100$ machine cycles. Then $N \times T_{\min} = 6 \times 100 = 600$ machine cycles. The allowable values for NT are then 601–1200 machine cycles, a total of 599 different values.

The fastest way to "calculate" f would be with a 599-byte look-up table. This method has the added advantage that nonlinearities in the transfer function between frequency and measurand can be built into the table. Look-up tables are facilitated in the 8051 by the `MOVC A, @A + PC`, and `MOVC A, @A + DPTR` instructions. `DPTR` is a 16-bit "data pointer" register in the 8051. Its two bytes can be individually addressed as `DPL` (low byte) and `DPH` (high byte).

In the example under discussion, it will be necessary to load `DPTR` with the address of the first byte of the look-up table, less 601, plus the 2-byte value of NT . The software that accesses the table might then take the following form:

TABLE EQU (address of first table entry)

```

DIVIDE ROUTINE
This divide routine calculates
    numerator
    quotient
    denominator
in which numerator and denominator are integers and
numerator / denominator = Quotient is of the form
    quotient = q0 q1 q2 q3 ... qN
where qN is the coefficient of 2(n-1). The procedure is

```

```

numerator = 2 * numerator
while n = N+1 do
    if numerator > denominator then qN = 0 else qN = 1
    if qN = 0 then numerator = 2 * numerator
    else numerator = 2 * numerator - denominator
    increment n
end while

```

Fig. 1. A divide algorithm.

```

MOV  A,#LOW(TABLE-601)
ADD  A,NTLO
MOV  DPL,A
MOV  A,#HIGH(TABLE-601)
ADDC A,NTHI
MOV  DPH,A
CLR  A
MOVC A,@A+DPTR

```

At this point the accumulator contains the 8-bit value of f .

It is perfectly reasonable to decide that a 599-byte look-up table is unwieldy. Its advantages are speed and built-in error correction. But a reasonably fast divide algorithm can be written to this specific purpose, making use of *a priori* knowledge about the sizes of the numbers that are involved in the computation. It helps to know that in this example the numerator is never going to be larger than 599 and the denominator is always greater than the numerator.

A complete discussion of divide routines is beyond the scope of this paper, but a suitable divide algorithm for this specific application is shown in Fig. 1. Reference [1] calls this the Restoring division algorithm. It is particularly well suited to the 8051, because "<" comparisons are greatly facilitated by the 8051's CJNE (compare and jump if not equal) instruction. CJNE A,B,rel , executes a relative jump if A does not equal B . More importantly to this application, the instruction sets the carry flag if $A < B$.

XI. ACCURACY AND RESOLUTION

The accuracy with which the 8051 will measure the frequency or period of the transducer signal depends on two things: the accuracy of the clock oscillator and variations in the interrupt response time.

Since the clock signal is normally generated by a crystal oscillator, the oscillator accuracy normally far exceeds the quantizing error inherent in the finite (n -bit) resolution.

As was previously mentioned, interrupt response time does not introduce an error into the measurement itself, but variations in the interrupt response time can. Interrupt response time in the 8051 can vary from 3 to 8 machine cycles, depending on what instruction is in progress at the time the interrupt is generated. This would represent an error of ± 5 counts in the measured value of NT during a period measurement. An error of ± 5 counts in NT does not necessarily translate to ± 5 LSB's in the final result, but it might still represent an error that exceeds the resolution.

In a direct frequency measurement variations in the interrupt response time would represent an error of $\pm 5 \mu s$ in the sample time.

If these kinds of errors are unacceptable there are ways to deal with them. In period measurements, if the duty cycle of the transducer is constant, the pulsewidth measurement technique, previously described, can be used. Its advantage is that it gates the timer off when the interrupt is generated, rather than when the interrupt is responded to.

In other cases one can simply increase the sample time above the minimum required to obtain the desired resolution. For example, if the measurement requires 8-bit resolution, one can design the software for an 11-bit resolution and truncate the result to 8 bits.

REFERENCES

- [1] Davio et al., *Digital Systems with Algorithm Implementation*. New York: Wiley, 1983.

ANALOG/DIGITAL PROCESSING WITH MICROCONTROLLERS

John Katausky, Ira Horden, Lionel Smith
Application Engineers
Intel Corp.
5000 W. Williams Field Road
Chandler, AZ 85224

Microcontrollers are rapidly becoming the backbone of silicon computing systems. From a technical standpoint, the most significant attribute, aside from the inclusion of RAM and ROM, that segregates a microcontroller from a microprocessor is I/O manipulation. In general, I/O manipulation is an intimate part of a microcontroller's architecture. The instruction set and architecture of a microcontroller allows the CPU to directly control the I/O facilities on the device. This is in direct contrast to a microprocessor where the I/O is essentially a "sea" of addresses and it is up to the hardware designer to place some type of I/O hardware in this I/O "sea". It should be obvious that simply adding ROM and RAM to a microprocessor WILL NOT create a microcontroller.

This intimate contact with I/O gives the microcontroller a distinct advantage over the microprocessor in applications that are I/O intensive. Microcontrollers can test, set, complement, or clear I/O port pins much faster than a microprocessor and they can also make decisions, based on the state of other hardware features, such as timer/counters with equal speed. This integration of I/O, in both hardware and software makes the microcontroller "ideal" for many types of intelligent instrumentation.

4K ROM/EPROM - 8K ROM ON 8052
128 BYTES OF RAM - 256 ON THE 8052
2-16 BIT TIMER/COUNTERS - 3 ON THE 8052
FULL DUPLEX UART
5 VECTORED INTERRUPTS - 6 ON THE 8052
4 REGISTER BANKS
BIT MANIPULATION (BOOLEAN PROCESSOR)
32 DIRECTLY ADDRESSABLE I/O PINS
MULTIPLY AND DIVIDE INSTRUCTIONS
SUPPORTS 64K OR RAM AND ROM-128K TOTAL

TABLE 1. A BRIEF LISTING OF THE MCS-51'S FEATURE SET.

Intel's MCS-51 series of microcontrollers contain many features that can be integrated directly into many types of instruments. TABLE 1 is a brief listing of these features. To illustrate the power of the 8051 this paper will elaborate on two techniques for performing analog to digital (A to D) conversion. Both of these examples assume that some additional hardware is attached to the I/O pins of the 8051.

S/A CONVERSION TECHNIQUES

Successive approximation analog to digital conversion involves a "binary search" of an unknown voltage relative to a "fixed" known reference. The reference is selectively divided by multiples of two until the desired accuracy is reached. Figure 1 is a flowchart of a successive approximation converter. This technique usually requires a digital to analog converter to divide the reference voltage and a voltage comparator to compare the unknown voltage to the "divided" reference. Digital to analog converters and voltage comparators are readily available and relatively inexpensive. A block diagram of an 8051 based A to D converter is shown in Figure 2.

Many industrial A to D converters require 12 bits of accuracy. A 12 bit converter provides good "dynamic range" and is capable of resolving 1 part in 4096. If the applied input voltage ranges from 0 to 10 Volts, a 12 bit converter can resolve 2.4 millivolts within this range. The theoretical accuracy of a 12 bit converter is .024% +/- 1/2 least significant bit.

The power of the 8051 in this type of application is best revealed by examining the software required to implement the successive approximation algorithm. The routine for the 8051 is shown in Table 2.

The execution times given assume a 12 Mhz crystal. Compare this to the following routine which is a 4 Mhz Z-80

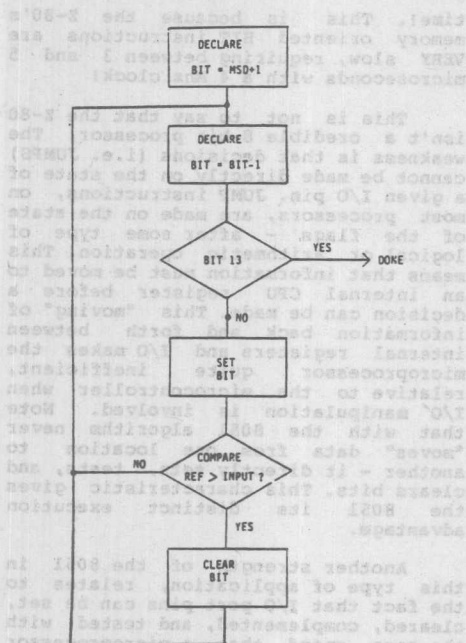


FIGURE 1. SUCCESSIVE APPROXIMATION
CONVERSION ALGORITHM

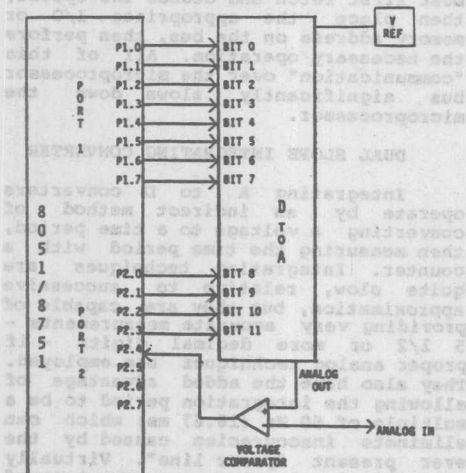


FIGURE 2. BLOCK DIAGRAM OF SUCCESSIVE APPROXIMATION A TO D CONVERTER

TABLE 2. SUCCESSIVE APPROXIMATION
ROUTINE FOR THE 8051.

INSTRUCTION	BYTES	TIME
; CLEAR PORT PINS		
MOV P1, #0	3	2
ANL P2, #0F0H	3	2
; START CONVERSION		
SETB P2.3	2	1
JNB P2.4, L1	3	2
CLR P2.3	2	1
L1: SETB P2.2	2	1
JNB P2.4, L2	3	2
CLR P2.2	2	1
L2: SETB P2.1	2	1
JNB P2.4, L3	3	2
CLR P2.1	2	1
L3: SETB P2.0	2	1
JNB P2.4, L4	3	2
CLR P2.0	2	1
L4: SETB P1.7	2	1
JNB P2.4, L5	3	2
CLR P1.7	2	1
L5: SETB P1.6	2	1
JNB P2.4, L6	3	2
CLR P1.6	2	1
L6: SETB P1.5	2	1
JNB P2.4, L7	3	2
CLR P1.5	2	1
L7: SETB P1.4	2	1
JNB P2.4, L8	3	2
CLR P1.4	2	1
L8: SETB P1.3	2	1
JNB P2.4, L9	3	2
CLR P1.3	2	1
L9: SETB P1.2	2	1
JNB P2.4, L10	3	2
CLR P1.2	2	1
L10: SETB P1.1	2	1
JNB P2.4, L11	3	2
CLR P1.1	2	1
L11: SETB P1.0	2	1
JNB P2.4, L12	3	2
CLR P1.0	2	1
; CONVERSION COMPLETE		
L12: CONVERSION COMPLETE		
TOTAL	90	46 US

NOTE: TIMING IS TYPICAL
WORST CASE = 52 US
BEST CASE = 40 US

executing the same algorithm with the D to A hardware attached to an I/O port is shown in Table 3 (assume that all bits on PORT3 are grounded, except the comparator input).

TABLE 3. SUCCESSIVE APPROXIMATION ROUTINE FOR THE Z-80.

INSTRUCTION	BYTES	TIME
; CLEAR PORT PINS		
LD A, 0	2	1.75
OUT (PORT1), A	2	2.75
OUT (PORT2), A	2	2.75
; START CONVERSION		
LD A, 08H	2	1.75
OUT (PORT2), A	2	2.75
IN A, (PORT3)	2	2.75
OR A	1	1.00
IN A, (PORT2)	2	2.75
JP Z, L1	3	2.50
AND 0F7H	2	1.75
L1: OR 04H	2	1.75
OUT (PORT2), A	2	2.75
IN A, (PORT3)	2	2.75
OR A	1	1.00
IN A, (PORT2)	2	2.75
JP Z, L2	3	2.50
AND 0FBH	2	1.75
L2: OR 02H	2	1.75
REPEAT BETWEEN L1 AND L2 10 MORE TIMES AND SET/RESET THE APPROPRIATE I/O BITS		

TOTAL 179 180 US

AGAIN TIMING IS TYPICAL
WORST CASE = 190.25 US
BEST CASE = 169.25 US

One may argue that by "memory mapping" the Z-80's I/O ports the execution time could be enhanced because the user could take advantage of the Z-80's SET and RESET memory BIT instructions. In reality, a few bytes of memory are saved, but very little

time! This is because the Z-80's memory oriented BIT instructions are VERY slow, requiring between 3 and 5 microseconds with a 4 Mhz clock!

This is not to say that the Z-80 isn't a credible 8-bit processor. The weakness is that decisions (i.e. JUMPS) cannot be made directly on the state of a given I/O pin. JUMP instructions, on most processors, are made on the state of the flags - after some type of logical or arithmetic operation! This means that information must be moved to an internal CPU register before a decision can be made. This "moving" of information back and forth between internal registers and I/O makes the microprocessor quite inefficient, relative to the microcontroller when I/O manipulation is involved. Note that with the 8051 algorithm never "moves" data from one location to another - it directly sets, tests, and clears bits. This characteristic gives the 8051 its distinct execution advantage.

Another strength of the 8051 in this type of application, relates to the fact that I/O port pins can be set, cleared, complemented, and tested with the same speed that a microprocessor can act on it's internal registers. Note that the 8051 takes only 1 microsecond to fetch an opcode and set or clear a port pin. A microprocessor must first fetch and decode the opcode, then place the appropriate I/O or memory address on the bus, then perform the necessary operation. All of this "communication" over the microprocessor bus significantly slows down the microprocessor.

DUAL SLOPE INTEGRATING CONVERTER

Integrating A to D converters operate by an indirect method of converting a voltage to a time period, then measuring the time period with a counter. Integrating techniques are quite slow, relative to successive approximation, but they are capable of providing very accurate measurements - 5 1/2 or more decimal digits - if proper analog techniques are employed. They also have the added advantage of allowing the integration period to be a multiple of 60 Hz (16.67 ms) which can eliminate inaccuracies caused by the ever present "power line". Virtually all digital voltmeters use some type of integrating technique. Figure 3 is a block diagram of a typical integrating

A to D converter. and hardware support. An assembler, and a high level language, presently, the 8051 is available in a technology "library" 8085, 8086, 8088, 8089, and 8090, depending on your individual application you can have it your way.

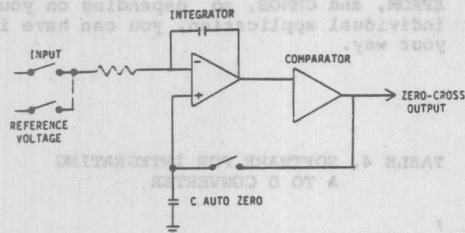


FIGURE 3. INTEGRATING A TO D CONVERTER

Figures 4A, 4B, and 4C show the three typical phases involved in the dual slope technique. Figure 4A illustrates the auto-zero phase. In this phase the integrating "loop" is closed and the offset of the analog integrator is accumulated in C auto zero. In Figure 4B, the input switch is closed and the integrator integrates the input voltage for a fixed time period T_1 . In figure 4C, the reference switch is closed and the integrator integrates the reference voltage until the comparator senses a zero crossing condition. The time it takes for this phase to occur is directly proportional to the amplitude of the input voltage. Additional circuitry can be added to determine the polarity of the input voltage, then switch in a reference of opposite polarity, but the basic technique remains the same.

The 8051 is an ideal controller for an intelligent integrating A to D system. The 16 bit timer/counters can provide better than $4 \frac{1}{2}$ decimal digits of accuracy, the serial port can be used to transmit the analog reading to a printer or another processor, the CPU can be interrupted by the 60 Hz line so conversions can start at precise intervals, and software can be used to calculate and save average, peak, or RMS readings.

Another "nice" benefit of this type of converter is that very few I/O port pins are required to control the A to D hardware, so opto-isolators can be used to completely isolate the 8051

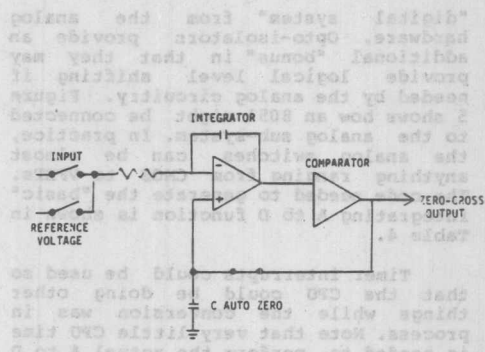


FIGURE 4A. AUTO ZERO PHASE

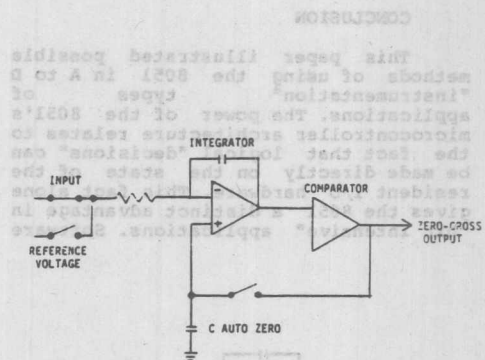


FIGURE 4B. INPUT INTEGRATION PHASE

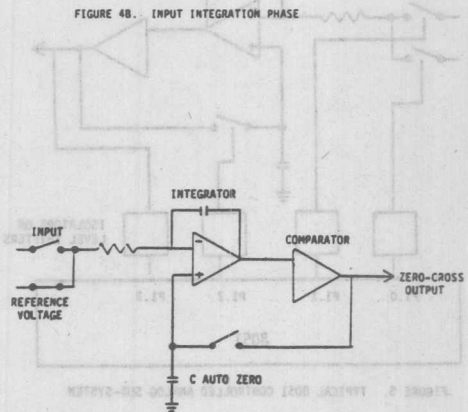


FIGURE 4C. REFERENCE INTEGRATION PHASE

Timer interrupts could be used so that the CPU could be doing other things while the conversion was in process. Note that very little CPU time is needed to perform the actual A to D function.

This paper illustrated possible methods of using the 8051 in A to "instrumentation" types of applications. The power of the 8051's microcontroller architecture relates to the fact that logical "decisions" can be made directly on the state of the resident I/O hardware. This fact alone gives the 8051 a distinct advantage in "bit intensive" applications. Software



TABLE 4. SOFTWARE FOR INTEGRATING
A TO D CONVERTER

ASIC Family Application Note **3** & Article Reprint

ASIC Family Application Note & Article Reprint

NOTE

PAGE	CONTENTS
3-3	INTRODUCTION
3-3	80C51-BASED ASIC CORE
3-4	RECONSTRUCTION OF STANDARD PRODUCT I/O
3-5	MULTIPLE SOURCES OF RESET
3-5	I/O EXPANSION WITH THE 80C51-BASED CORE
3-7	ON-CHIP CLOCK GENERATION
3-8	SHARING THE TEST BUS
3-9	OBSERVING THE CONTENTS OF THE PROGRAM COUNTER
3-9	DESIGN EXAMPLE
3-11	REFERENCE DOCUMENTATION

USING INTEL'S ASIC CORE CELL TO EXPAND THE CAPABILITIES OF AN 80C51-BASED SYSTEM

July 1988

3

Using Intel's ASIC Core Cell to Expand the Capabilities of an 80C51-Based System

MATT TOWNSEND
CPO TECHNICAL MARKETING MANAGER

Order Number: 240230-001

USING INTEL'S ASIC CORE CELL TO EXPAND THE CAPABILITIES OF AN 80C51-BASED SYSTEM

CONTENTS

PAGE

INTRODUCTION	3-3
80C51-BASED ASIC CORE	3-3
RECONSTRUCTION OF STANDARD PRODUCT I/O	3-4
MULTIPLE SOURCES OF RESET	3-5
I/O EXPANSION WITH THE 80C51-BASED CORE	3-5
ON-CHIP CLOCK GENERATION	3-7
SHARING THE TEST BUS	3-8
OBSERVING THE CONTENTS OF THE PROGRAM COUNTER	3-9
DESIGN EXAMPLE	3-9
REFERENCE DOCUMENTATION	3-11

July 1988

3

Using Intel's ASIC Core Cell to
Expand the Capabilities of an
80C51-Based System

MATT TOWNSEND
CPO TECHNICAL MARKETING MANAGER

Order Number 540530-001

INTRODUCTION

Intel's new ASIC family of microcontroller core cells extends the capability of the MCS®-51 product, and allows the ASIC designer more flexibility than the popular microcontroller product. This note will discuss many of the new design possibilities inherent to the 80C51 cell-based controller. This family of cells is available with a variety of RAM and ROM configurations.

Cell Name	ROM	RAM
UC5100	No ROM	128 Bytes RAM
UC5104	4K ROM	128 Bytes RAM
UC5108	8K ROM	128 Bytes RAM
UC5116	16K ROM	128 Bytes RAM
UC5200	No ROM	256 Bytes RAM
UC5204	4K ROM	256 Bytes RAM
UC5208	8K ROM	256 Bytes RAM
UC5216	16K ROM	256 Bytes RAM

Other documentation will address Intel's ASIC design environment (see reference section).

The 80C51-based ASIC cell is part of a family of cell-based functions based on popular Intel standard products. Members of the 82Cxx microprocessor support peripheral family (SP8254, SP8237, SP8259, SP8284, SP82284, SP8288 and SP82288) are also available as library elements. The standard product ASIC cores are supported by a library of over 150 logic cells, representing a broad range of SSI, MSI, and I/O functions. Another class of cell library elements is designated Special Functions. These cells are predefined complex functions such as RAM, Serial I/O, A/D Converter, and a Voltage Comparator. The Special Function and general logic element cells can also be used without a standard product core in the ASIC design. Any of the available 80C51-based cores can be integrated with logic complexities up to 5000 gates.

80C51-BASED ASIC CORE

Although the 80C51-based core is functionally identical to the standard 80C51BH microcontroller, its use as a cell in the ASIC library allows more flexibility in system design and partitioning.

Figure 1 depicts the difference between the standard pinout of the MCS-51 family and the ASIC core. In order to understand the enhancements (in an applications sense) made to the core it is useful to compare its connections to the pinout of the standard product.

The MCS-51 family embodies a very powerful architecture. While it was intended as a "single chip solution"

its addressing modes, clean bus interface, on-chip peripherals, and code efficient instruction set operations make it well suited to processor-like applications as well. For processor applications, a designer forgoes many of the "single chip" features in favor of the high performance CPU functions of this architecture.

In order to fit the MCS-51 family microcontrollers into an economical forty lead DIP or forty-four lead PLCC package, Intel designed the standard product with many of the device's functions sharing pins. The microcontroller designer must compare necessary functions against the economics and performance required for a given design. If external memory or memory mapped I/O is required, then the use of the port 0 function is not available. If the memory address is beyond the 256 byte boundary defined by the AD0-7 Bus then all or part of the port 2 function is not available. Likewise, using peripheral functions like the counter input pins, serial I/O, and interrupts eliminates port 3 functions. While the MCS-51 family is one of the most popular microcontrollers ever introduced, this shared functionality hinders its use in many applications. For example, a "fully loaded" MCS-51-based design would generally leave only one 8-bit port (Port 1) for the application's I/O requirements.

The standard cell version of the 80C51 provides the designer with 116 signals for connection to application specific logic. These signals represent the full function set of the MCS-51 architecture and virtually eliminate any design trade-offs required to implement an application. Notice from Figure 1 that all of the I/O ports are separated from the other functions. In the design example, the I/O are separated into their respective inputs and outputs, leaving 32 inputs and 32 outputs for port connections into the application's logic. The most immediate impact of demultiplexing the I/O of the device is that much of the logic required to complete an application is eliminated. For example, when separating the address from the data on the AD-bus, an octal latch is required. For an 80C51-based core application, the designer uses the A0-7 bus directly, thus saving approximately 100 gates. The fact that the 80C51-based core has so many connections available does not mean an application will be forced into higher pin count packages. A 80C51-based ASIC can implement many system functions more economically than a discrete implementation. The design example illustrates a system with over 280 interconnects that can be integrated into one ASIC device. This application note will illustrate the less obvious ways in which the core can be used.

The illustrations shown in this note are independent of the workstation platform used to implement the design.

Intel provides the complete test vectors necessary to test the 80C51-based ASIC core, which have been derived from the standard product 80C51 test vector set.

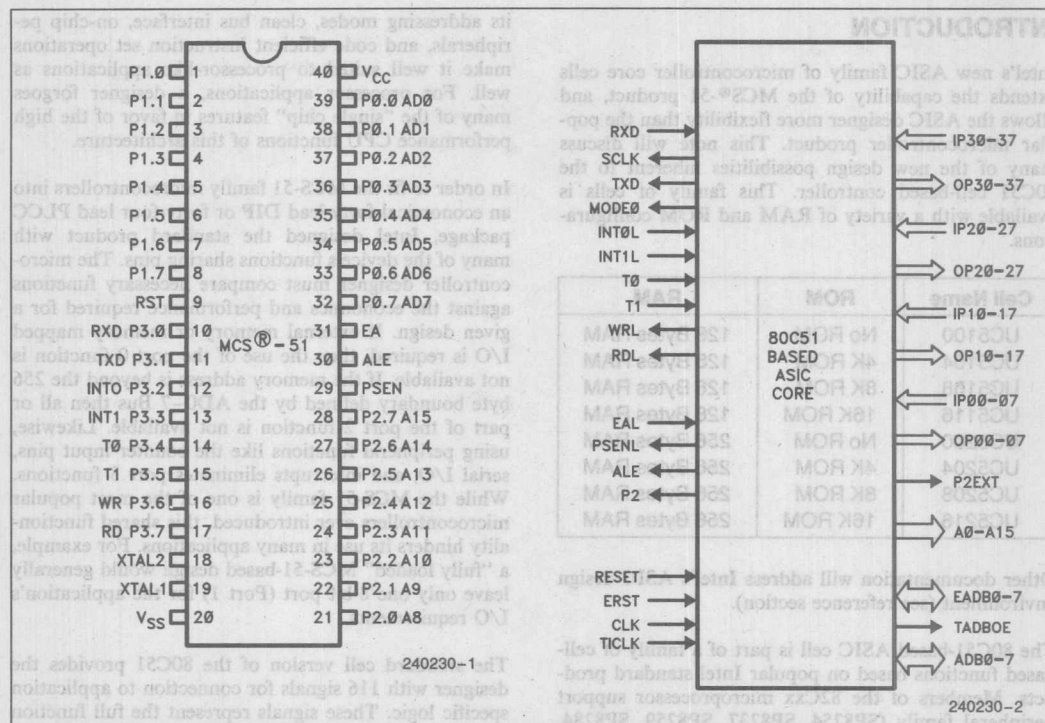


Figure 1. Comparing 80C51 Pin Assignments to Core Connections

RECONSTRUCTION OF STANDARD PRODUCT I/O

When designing the 80C51-based ASIC core, Intel removed the pin multiplexers and I/O functions of the 80C51, and restructured them as companion cells. Companion cells allow the ASIC designer to reconfig-

ure the ASIC cell to function exactly like the standard product. Alternatively, the designer can choose to reconstruct a subset of the standard product I/O or select no reconstruction at all. Consult the references for more information about the use and function of Intel's companion cells.

Other documentation will address later design (reference section). The ASIC cell is part of the standard product core in the ASIC design. Any of the available 80C51-based cores can be integrated with logic components up to 3000 gates.

80C51-BASED ASIC CORE

Although the 80C51-based core is functionally identical to the standard 80C51BH microcontroller, its use as a cell in the ASIC library allows more flexibility in system design and partitioning.

Figure 1 depicts the difference between the standard product of the MCS-51 family and the ASIC core. In order to understand the enhancements (in an applications sense) made to the core it is useful to compare its connections to the pinout of the standard product.

The MCS-51 family embodies a very powerful architecture. While it was intended as a "single chip solution,"

The illustrations shown in this note are independent of the workstation platform used to implement the design.

Intel provides the complete test vectors necessary to test the 80C51-based ASIC core, which have been derived from the standard product 80C51 test vector set.

MULTIPLE SOURCES OF RESET

Key to the 80C51's core-isolation test method is the ability to put the core into a condition that can verify the processor without the user's logic affecting the test. ERST is vital to controlling the ability to put the core based ASIC into test mode. It must be brought directly from the core to a package pin. Interface is via the PRESET companion cell. Because a dedicated reset pin may be restrictive in many applications, a second reset connection, RESET, has been included.

Including this second reset connection allows the designer to simplify the overall ASIC design. Many applications require two sources of reset, usually a power-on-clear with a watchdog timer. Previously, the designer was faced with "ORing" an RC time constant circuit with the timer logic, resulting in an implementation which was not straightforward or cost effective. Figure 2 shows an 80C51-based ASIC implementation.

A system reset, in many designs, employs an active low logic level. Since the 80C51's reset requires an active high level, there is usually an inverter in the path to the microcontroller. It was mentioned earlier in this section that ERST must be brought directly from the core to a package pin. This is not entirely true; the inclusion of the inverter is allowed.

I/O EXPANSION WITH THE 80C51-BASED CORE

For the standard MCS-51 product, the need for I/O expansion is often due to the need for external memory and/or port expansion. The designer's use of the on-chip peripherals (eg. Serial I/O or Interrupts) often leaves only Port 1 intact.

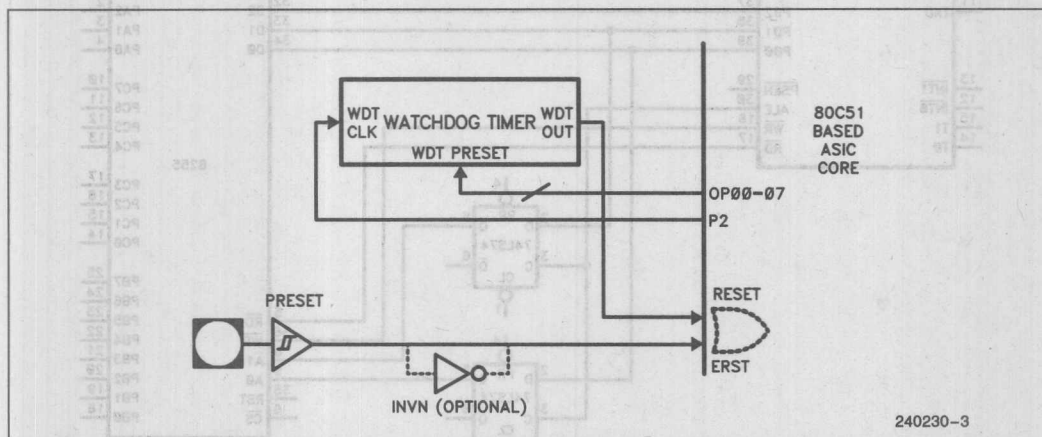


Figure 2. Multiple Sources of Reset for 80C51-Based ASIC Core

240230-3

Figure 3. Simple MCS-51 I/O Expansion with 8255

Figure 3 depicts one case where the 80C51 can gain an expanded set of I/O ports. In addition to requiring additional package pins, this implementation would require more power supply capacity and passive components (opamp capacitors) than would be necessary if the I/O expansion were to be included on-chip with the microcontroller. Not only is IC board size decreased, but the overall system reliability increases with the ASIC solution. In addition, the 8255 port expansion being a highly flexible device requires software to configure the device to the application.

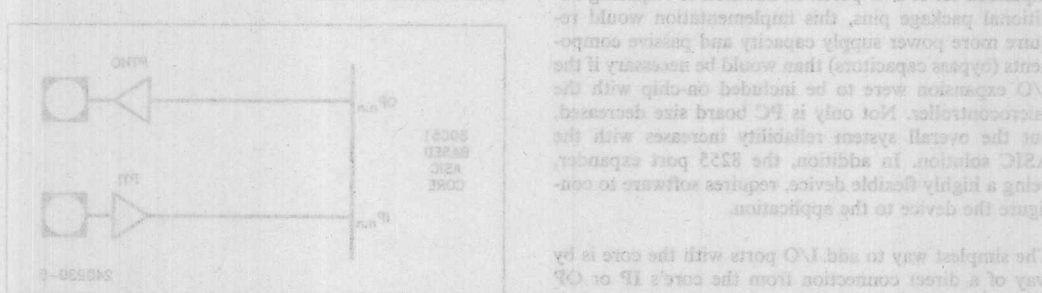


Figure 4. Direct Port Connections to ASIC Package Pins

The simplest way to add I/O ports with the core is by way of a direct connection from the core's IP or OP signals through I/O functions selected from the cell library and connected to package pins. See Figure 4. In

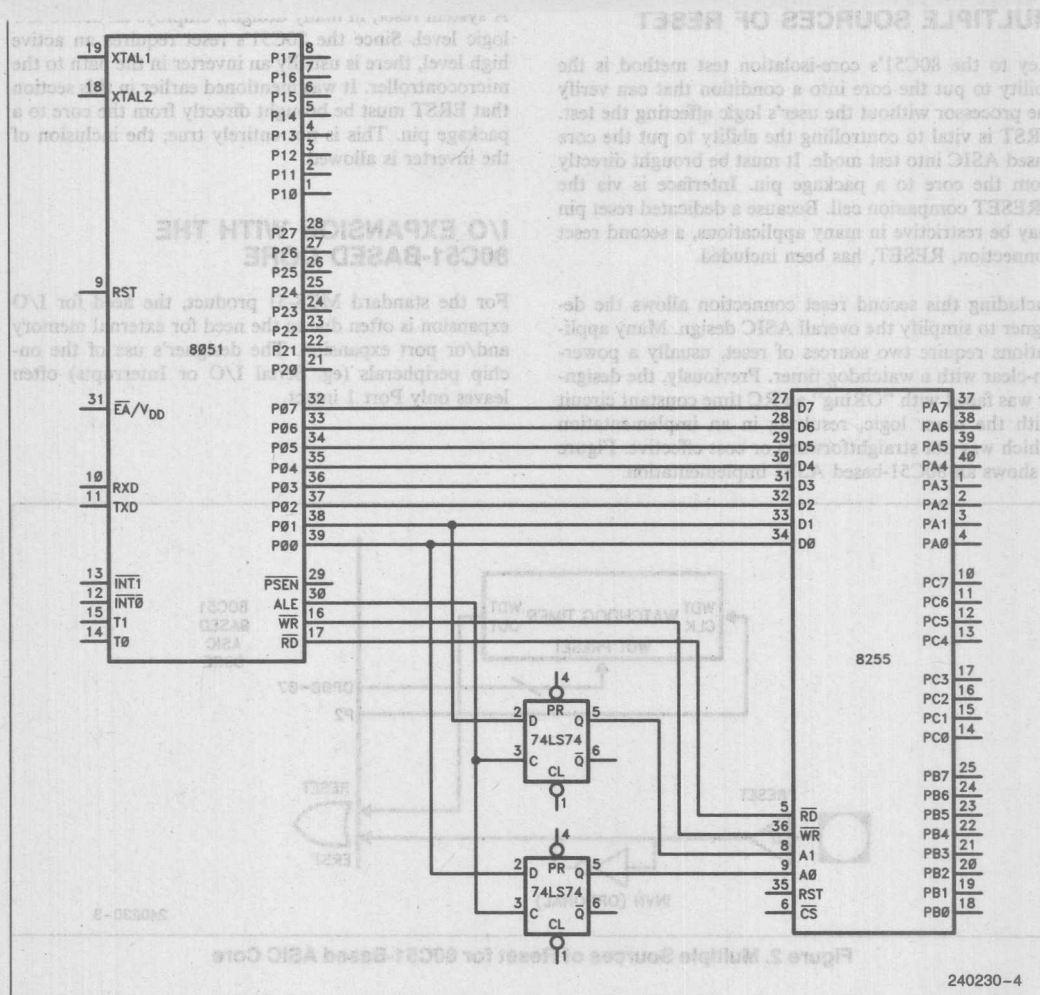


Figure 3. Simple MCS®-51 I/O Expansion with 8255

Figure 3 depicts one case where the 80C51 can gain an expanded set of I/O ports. In addition to requiring additional package pins, this implementation would require more power supply capacity and passive components (bypass capacitors) than would be necessary if the I/O expansion were to be included on-chip with the microcontroller. Not only is PC board size decreased, but the overall system reliability increases with the ASIC solution. In addition, the 8255 port expander, being a highly flexible device, requires software to configure the device to the application.

The simplest way to add I/O ports with the core is by way of a direct connection from the core's IP or OP signals through I/O functions selected from the cell library and connected to package pins. See Figure 4. In

this example, decoding is very simple and the component count is minimal.

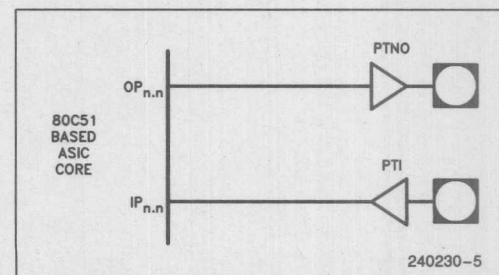


Figure 4. Direct Port Connections to ASIC Package Pins

This technique represents the most silicon and program code efficient way to implement I/O with the core. Program code is composed of MOV operations rather than the MOVX needed for any Memory mapped I/O implementations. Logical operations to the port (ANL, ORL, XRL) can still be used. It is not necessary to update or maintain indirect pointers. Since quasi-bidirectional cells are not used, it is not necessary to set a "1" to the port in order to set these cells. Eliminating MOVX and port setup operations could result in significant codespace savings.

The drawback to the direct approach is that program code written for the 80C51 using memory mapped I/O is not directly transportable to the core design. In most cases, however, implementing I/O expansion that allows code transportability is a simple task with a 80C51-based ASIC.

Most support peripherals are designed to be configurable for many different operating conditions. This is certainly true for a device like the 8255 as well; the port signals can be programmed as inputs, outputs, or bidirectional. In most applications the peripheral's setup is never changed after initialization. Port pins are set to either input, output or I/O. The peripheral's configuration is most often "set up" with data fields sent to a configuration or command register. This register is located at one of the peripheral's selectable addresses. For a cell-based implementation where code transportability is required, recreating an 8255-like function is straightforward. Since all setup information is written to one register and setup is not required because the port signal directions are fixed, that one register can

become a "bit bucket". Figure 5 shows an example with one 8-bit input port, and one 8-bit output port, both memory-mapped. Note that while the 8255 contains three 8-bit ports, an ASIC can be implemented with the exact amount of functionality desired.

Implementing the full function set of the 8255 would result in an increased gate count (550 gates) included on the 80C51-based ASIC. While 550 gates can easily be included on the same silicon chip, implementing the "exact functionality" version using elements from the cell library would consume only 100 gates.

ON-CHIP CLOCK GENERATION

In many designs, the built-in crystal oscillator of the 80C51 is not utilized because the clock signal must be used for other system functions. However, the clock to the 80C51 still must be generated and driven into the X1/X2 pins. A clock generator is required somewhere in the system and is also used to clock many of the system functions surrounding the microcontroller.

For 80C51-based ASIC designs, all clocking functions, including the clock generator, can be brought on-chip. The advantages to doing so include enhanced reliability and a less costly, more noise free design. Clock generation is accomplished by the companion cell POSC (or POSC2 for frequencies between 16 MHz and 38 MHz) which can be used to drive the T1CLK input connection to the core. The POSC output can be sent to user defined logic configured to generate other necessary

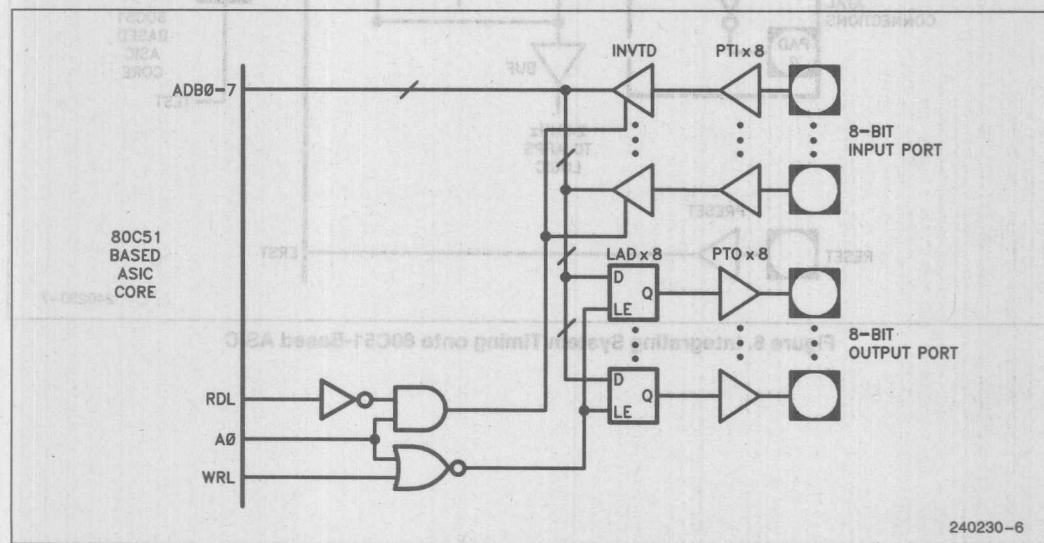


Figure 5. I/O Mapped Like Figure 3's 8255

Note that signal P2 is shown being used for the application's clocks. It is sent to other logic in the ASIC and to a package pin as well.

In order for Automatic Test Equipment (ATE) to exercise the 80C51-based ASIC, the bus EADB must be brought directly to package pins. A specially designed I/O cell, PADB, must be directly connected to the EADB bus to ensure testability of the core as well as the user's logic.

Intel supplies all test programs required to completely test the ASIC core cell. Designers are required to supply test vectors that exercise their unique logic only.



OBSERVING THE CONTENTS OF THE PROGRAM COUNTER

The 80C51-based ASIC core connections A0–A15 always display the contents of the program counter (except in the case of MOVX instructions.) This feature allows another level of real time control by monitoring instruction events within the core. By attaching comparator circuitry to the program counter contents, signals can be generated to depict events within the program. Figure 7 shows such a circuit.

The discussions in this note are not intended to be an exhaustive summary of the range of design possibilities available to the ASIC designer. Rather, it is hoped that it encourages the thought process toward even more innovative uses.

The following is an example of an actual system problem and how it was resolved using a 80C51-based ASIC. The example utilizes many of the techniques discussed above.

DESIGN EXAMPLE

Figure 8 shows a typical MCS-51-based design, which includes a port expander, timer/counter chip, a high speed event counter and a low-cost EPROM containing stable code. In this application, the 8031 controls a system based on numerous timed events. Many high speed clocks are involved, making for a potentially noisy environment, and a watchdog timer has been included to provide for soft recoveries if the microprocessor program flow is upset. The watchdog circuitry is shown as a high level block.

The design must take an accurate sample of events designated at the EVENT input. The 16-bit count is read and processed under a timed interrupt designated by and generated from one of the 8254 Timers. The counter chain must be clocked at 24 MHz in order for unique and accurate event samples to occur.

Another 8254 counter is programmed for single-shot mode to provide for a strobe window for some circuitry external to the PCB assembly. An 8-bit parallel data

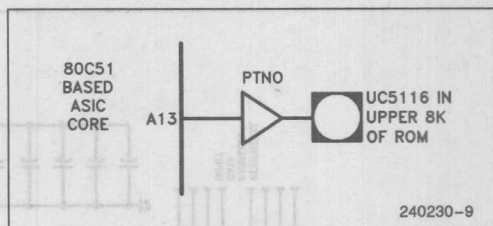


Figure 7. Signal which Designates when Program Execution is in Upper 8K ROM

input is required. The system must control peripherals external to this main assembly, resulting in a requirement for address decoder selection.

Note that adequate bypass capacitors are required due to the clock speeds and the high number of pin connections. (There are 272 pins, not including the WDT and Oscillator Blocks.) A multilayer PCB is also required to compensate for the amount of wires needed to connect all the components.

Figure 9 depicts the 80C51-based ASIC solution for the design in Figure 8. Note that all of the circuitry in Figure 8 is included on a single ASIC chip. Rather than use memory-mapped I/O, the design has been converted to use the core's direct ports. Note that the 8255 function has been removed and instead the UC5116 port connections are used. Some minor software changes are required, and signals required to access off-chip program memory have been provided. This figure does not show all test pin requirements; however, no additional package pins will be required. For this example, the designer could begin production runs with an EPROM. Once the application code stabilized, it could be developed and submitted to Intel for incorporation into the core. In this example, most of the high speed signals are contained within the ASIC, making the watchdog timer unnecessary. If needed, the overall cost of including it on the ASIC (= 500 gates) makes the functions relatively inexpensive to keep.

Overall system pin requirements have decreased as well. This 80C51-based ASIC can be produced using a 68-pin PLCC which may reduce the bypass capacitor requirements as well as the need for a multilayer PCB.

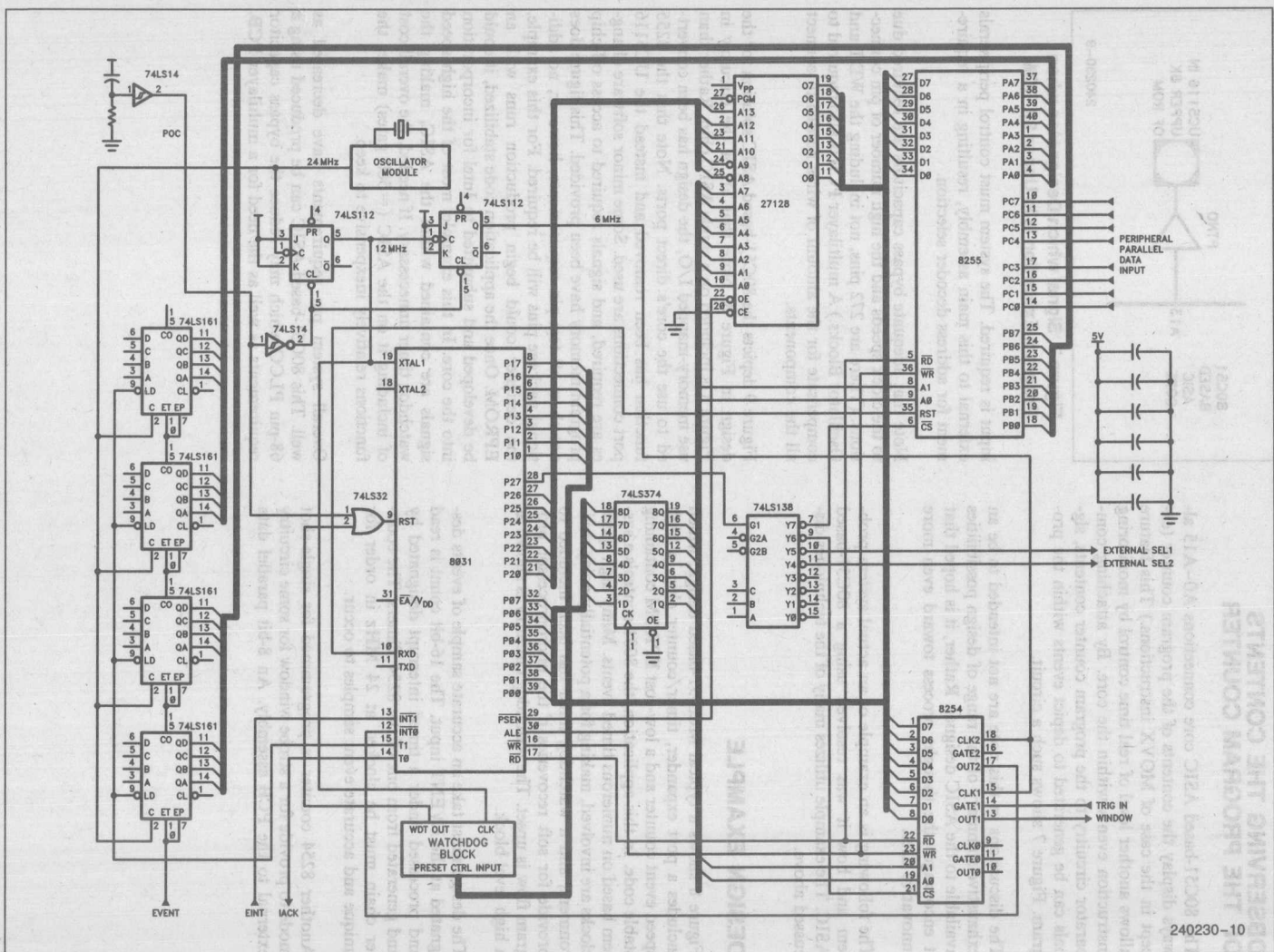


Figure 8. Typical MCS-51 Design, Which Includes a Port Expander, Timer/Counter Chip, a High-Speed Event Counter, and a Low-Cost EPROM

Comparing the two solutions:

	Discrete	80C51-ASIC
Component Count	~ 25	1
Minimum PCB Layers	3	1
System Reliability	Medium	High
Power Supply Current	~ 3A	~ 30 mA

REFERENCE DOCUMENTATION

Order Number	Description
231816	— Introduction to Cell-Based Design
83002	— Cell-Based Design—Daisy Environment
830000	— Cell-Based Design—Mentor Environment
210918	— Embedded Controller Handbook
270535	— Embedded Control Applications

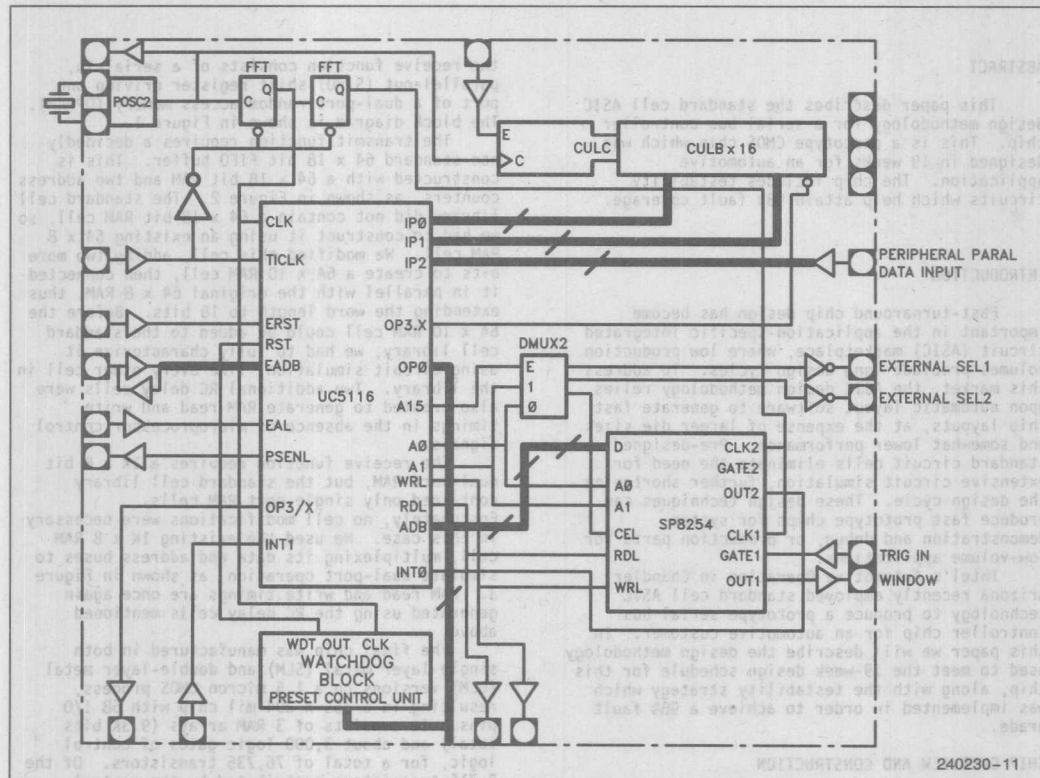


Figure 9. 80C51-Based ASIC Solutions

Chip for Serial Bus Control

Don Ellis and Shailesh Trivedi

Intel Corp.

Chandler, AZ

Power Supply Current	~3A	~30 mA
System Reliability	Medium	High
PCB Layers	3	1
Die Size	~25	~1
Process	80C8-ASIC	80C8-ASIC

ABSTRACT

This paper describes the standard cell ASIC design methodology for a serial bus controller chip. This is a prototype CMOS chip which was designed in 19 weeks for an automotive application. The chip includes testability circuits which help attain 98% fault coverage.

INTRODUCTION

Fast-turnaround chip design has become important in the application-specific integrated circuit (ASIC) marketplace, where low production volumes preclude long design cycles. To address this market, the ASIC design methodology relies upon automatic layout software to generate fast chip layouts, at the expense of larger die sizes and somewhat lower performance. Pre-designed standard circuit cells eliminate the need for extensive circuit simulation, further shortening the design cycle. These design techniques can produce fast prototype chips for system demonstration and debug, or production parts for low-volume applications.

Intel's Automotive Operation in Chandler, Arizona recently employed standard cell ASIC technology to produce a prototype serial bus controller chip for an automotive customer. In this paper we will describe the design methodology used to meet the 19-week design schedule for this chip, along with the testability strategy which was implemented in order to achieve a 98% fault grade.

CHIP OVERVIEW AND CONSTRUCTION

The serial bus controller is a standard cell CMOS chip that interfaces a microprocessor to a serial communication bus in an automobile. The chip performs both transmit and receive functions. The transmit function consists of a first-in, first-out (FIFO) data buffer feeding a parallel-in, serial-out (PISO) shift register, and

the receive function consists of a serial-in, parallel-out (SIPO) shift register driving one port of a dual-port random access memory (DPRAM). The block diagram is shown in Figure 1.

The transmit function requires a decidedly non-standard 64 x 18 bit FIFO buffer. This is constructed with a 64 x 18 bit RAM and two address counters, as shown in Figure 2. The standard cell library did not contain a 64 x 18 bit RAM cell, so we had to construct it using an existing 64 x 8 RAM cell. We modified this cell, adding two more bits to create a 64 x 10 RAM cell, then connected it in parallel with the original 64 x 8 RAM, thus extending the word length to 18 bits. Before the 64 x 10 RAM cell could be added to the standard cell library, we had to fully characterize it using circuit simulation, like every other cell in the library. Two additional RC delay cells were also created to generate RAM read and write timings in the absence of microprocessor control signals.

The receive function requires a 1K x 8 bit dual-port RAM, but the standard cell library contained only single-port RAM cells.

Fortunately, no cell modifications were necessary in this case. We used the existing 1K x 8 RAM cell, multiplexing its data and address buses to simulate dual-port operation, as shown in Figure 3. RAM read and write timings are once again generated using the RC delay cells mentioned above.

The final chip was manufactured in both single-layer metal (SLM) and double-layer metal (DLM) versions on a 1.5 micron CMOS process, resulting in a 355 x 294 mil chip with 68 I/O pins. It consists of 3 RAM arrays (9.3K bits total) and about 3,000 logic gates of control logic, for a total of 76,735 transistors. Of the 8,715 transistors contributed by the control logic, 11% belong to testability circuits which were added to increase the testability of the chip (i.e., shorten test program development time and tester run time). The testability strategy will be discussed later.

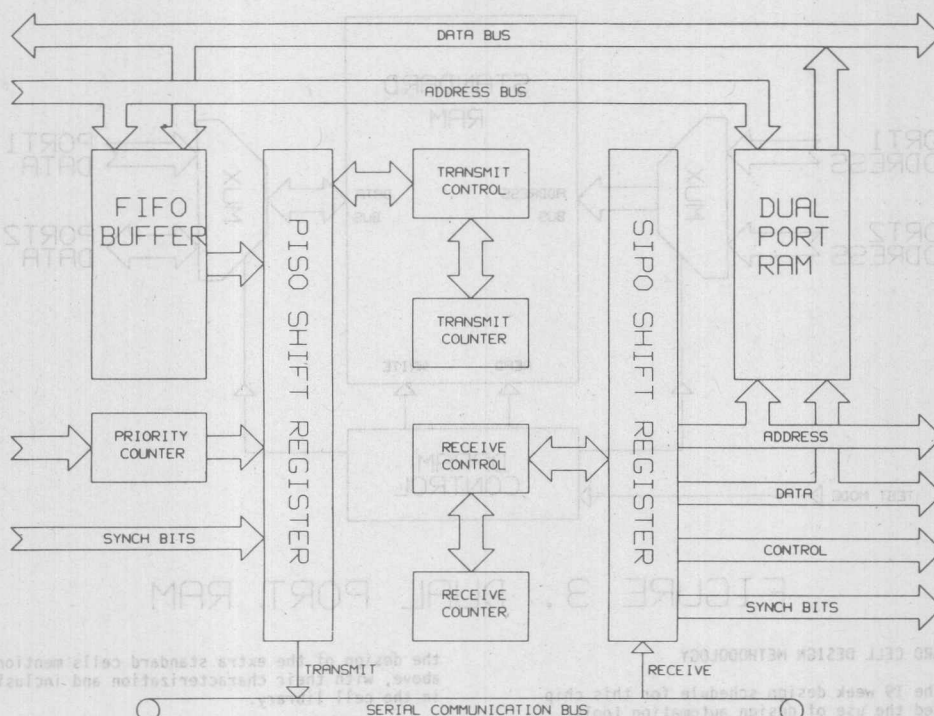


FIGURE 1. SERIAL BUS CONTROLLER BLOCK DIAGRAM

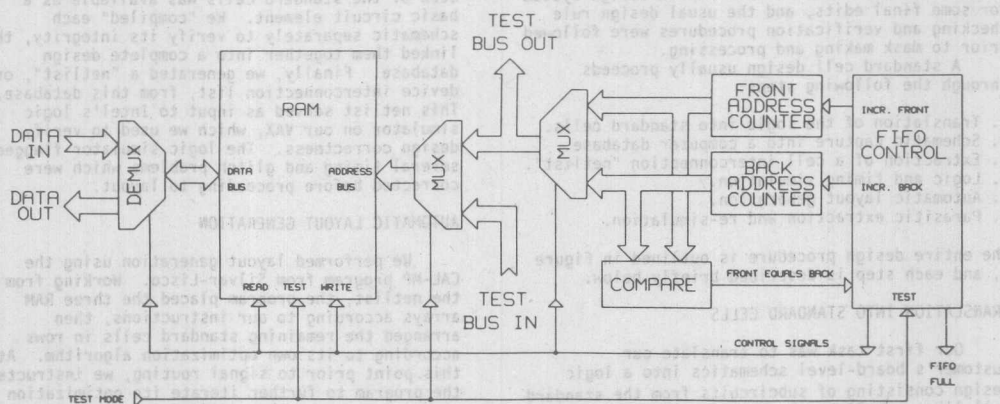


FIGURE 2. FIFO (FIRST-IN, FIRST-OUT) BUFFER

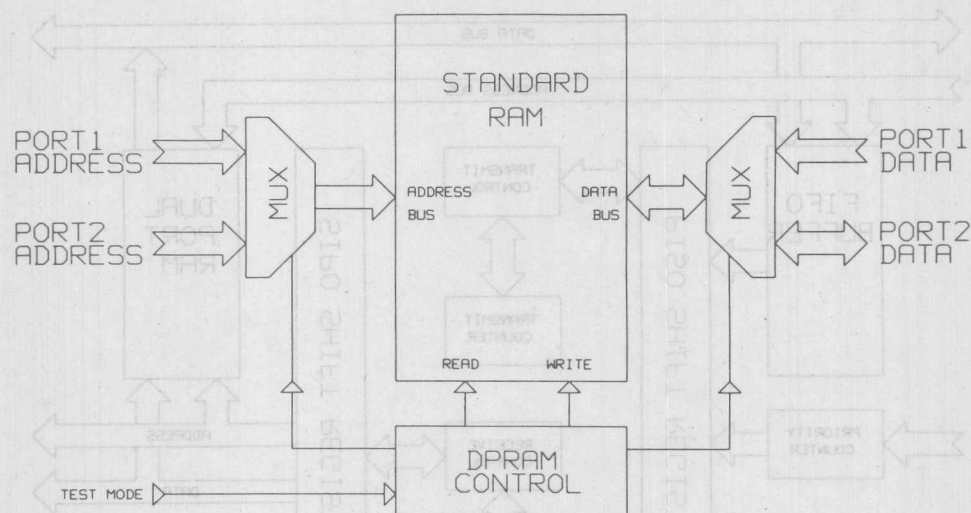


FIGURE 3. DUAL PORT RAM

STANDARD CELL DESIGN METHODOLOGY

The 19 week design schedule for this chip dictated the use of design automation tools. Since the chip included 3 large RAM arrays, gate arrays were impractical, so standard cells were used with automatic placement and routing software. The automatically generated layout was transferred into Intel's full custom design system for some final edits, and the usual design rule checking and verification procedures were followed prior to mask making and processing.

A standard cell design usually proceeds through the following steps:

1. Translation of the logic into standard cells.
2. Schematic capture into a computer database.
3. Extraction of a cell interconnection "netlist".
4. Logic and timing simulation.
5. Automatic layout generation.
6. Parasitic extraction and re-simulation.

The entire design procedure is outlined in Figure 4, and each step is described briefly below.

TRANSLATION INTO STANDARD CELLS

Our first task was to translate our customer's board-level schematics into a logic design consisting of subcircuits from the standard cell library. Since the customer's schematics referenced IC packages only, this involved the detailed design of the FIFO and DPRAM blocks (described above). A major part of the task was

the design of the extra standard cells mentioned above, with their characterization and inclusion in the cell library.

SCHEMATIC CAPTURE, NETLIST EXTRACTION, SIMULATION

We performed schematic capture on a Daisy Personal Logician (PC-AT based) workstation, where each of the standard cells was available as a basic circuit element. We "compiled" each schematic separately to verify its integrity, then linked them together into a complete design database. Finally, we generated a "netlist", or device interconnection list, from this database. This netlist served as input to Intel's logic simulator on our VAX, which we used to verify design correctness. The logic simulator flagged several timing and glitch problems which were corrected before proceeding to layout.

AUTOMATIC LAYOUT GENERATION

We performed layout generation using the CAL-MP program from Silvar-Lisco. Working from the netlist, the program placed the three RAM arrays according to our instructions, then arranged the remaining standard cells in rows according to its own optimization algorithm. At this point prior to signal routing, we instructed the program to further iterate its optimization steps, as we manually modified several cell placements from the graphics terminal. Once all cell placements were determined, the program performed signal and power routing automatically.

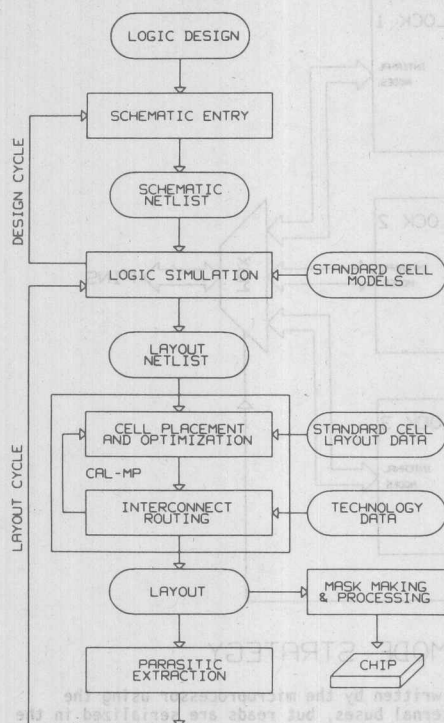


FIGURE 4. STANDARD CELL DESIGN FLOW

The CAL-MP program accepted layout constraints in a variety of forms. In addition to the netlist information, we defined pad placements and the number of standard cell rows, and constrained a few critical signals (such as clocks) to two vertical metal buses traversing the right and left sides of the chip. Furthermore, several unconstrained signals were assigned numerical "strengths" greater than the default of 1.0, which weighted their consideration in the optimization algorithm, tending to shorten them. We ultimately generated more than 20 layouts with widely varying signal strengths, until we were satisfied that very little further improvement was possible.

PARASITIC EXTRACTION

After a layout is generated, it must be proven to work in the presence of parasitic resistances and capacitances contributed by the signal interconnects. These parasitics are extracted from the layout and added to the netlist for a post-layout simulation cycle. In principle, each iterated layout should be re-simulated, but after about 10 layout generations we could easily

predict simulation performance from the raw parasitic values.

Because the first version of this chip was fabricated in single-layer metal with a great deal of polysilicon interconnection, parasitic series resistances were just as important in limiting performance as are parasitic capacitances. Unfortunately, series resistors are difficult to systematically insert into a netlist, so we had to simulate the resulting RC delays using Intel's circuit simulator. For the double-layer metal version, we could safely ignore series resistances since the metal sheet resistance is three orders of magnitude smaller than that of polysilicon.

DESIGN FOR TESTABILITY

Our test goal for this part was a 98% fault grade, and since this was a fast-turnaround project with little time for test program development, we included a variety of testability circuits on the chip. An added benefit to this approach was that the testability circuits simplified our debugging procedures. This strategy ultimately paid off, because we were able to quickly isolate and correct a RAM timing problem on the first silicon.

Since this chip has a relatively small node count, we adopted an "ad hoc" rather than "structured" testability strategy. This means that we added test circuits on a case by case basis to improve the controllability and observability of the overall chip, rather than implementing a scan path, a built-in self test, or some other more elaborate scheme. Ad hoc testability design is appropriate for small chips having relatively low transistor/pin ratios. This chip has 8,715 transistors (excluding those in the RAMs) versus 68 pins, for a transistor/pin ratio of 128. In contrast, Intel's 80386 microprocessor has 275,000 transistors versus 132 pins for a ratio of 2,083, clearly requiring structured testability techniques.

Two pins are allocated for test purposes, which are used to select among four modes: a normal operating mode, and three test modes. This test mode strategy is shown in Figure 5. The test modes are used to partition the chip into three isolated subcircuits to be tested independently. In each mode, signals with poor visibility internal to the active subcircuit are brought out to the pads, and the non-active subcircuits are turned off by disabling their clock inputs. The test program can then exercise the active circuit, with the goal of toggling each internal node for maximum fault coverage.

Eleven of the 28 chip inputs provide test inputs in the three test modes, and 16 of the 23 chip outputs serve double duty as test outputs. Although input pins can be connected to several internal test points in parallel (usually multiplexer inputs), only one signal at a time can drive an output pin. These outputs are multiplexed using three stages of 2:1 multiplexers (one for each mode), and the outputs are collected into a 16 bit "test bus" which circumnavigates the chip.

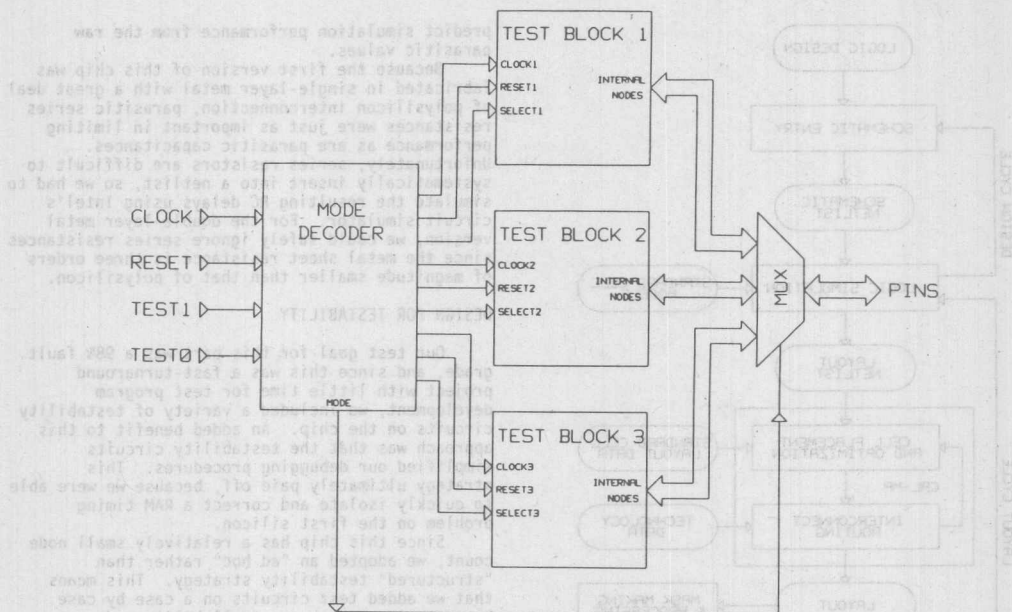


FIGURE 5. TEST MODE STRATEGY

RAM TESTABILITY

RAM testability is a special case, because a RAM is inherently fully testable provided its data and address buses are accessible, along with the necessary control signals. The difficulty here is that the RAMs are embedded in functional blocks which, especially in the FIFO, tends to disguise the inherent RAM accessibility.

Inside the DPRAM, the 1K x 8 RAM is read directly by the microprocessor using the external data and address buses, so observability is no problem. Writes, however, occur from the SIPO during serial reception. It would be particularly painful to test a 1K RAM using serial writes, so a modification was necessary to improve RAM controllability. In test mode, the address multiplexer is held to the address bus by overriding the select line, and a set of eight extra multiplexers were added to the data demultiplexer to allow bidirectional data flow into and out of the RAM. Thus, the SIPO circuit is completely bypassed in test mode.

The FIFO RAM is addressed by one of two counters in the operating mode, which presents a problem unless we are willing to accept sequential test addressing, or at least a very complex address setup procedure. The solution was to override the address bus with a multiplexer fed by input pin signals. The data bus presented the same problem as the DPRAM, but in reverse: data

is written by the microprocessor using the external buses, but reads are serialized in the PISO. We decided that serial output was acceptable in FIFO test mode, however, because the FIFO has only 64 locations to test (versus 1024 in the DPRAM), and the words are 18 bits long, which would require 18 extra multiplexers. Thus, for the FIFO data, we left well enough alone.

CONCLUSION

This serial bus controller chip was designed using ASIC techniques in a very short time, resulting in a quick prototype chip which our automotive customer could use to evaluate his system design in a timely manner. The inclusion of testability circuits further shortened the engineering debug time as well as the manufacturing test time. This project demonstrates that standard cell design is an attractive fast-turnaround methodology, and that a good testability strategy provides additional benefits which outweigh the extra design effort.

ACKNOWLEDGEMENT

The authors would like to thank Graham Tubbs, for guiding us through the maze of ASIC design tools, Dinesh Maheshwari and Keith Steele, who helped prepare our final layout for processing, and Mukund Patel and Magdiel Galan who helped us test and debug the final chip.

RUPI™ Application Notes

4

RUPITM Application Notes

4

4



APPLICATION NOTE

AP-281

September 1989

4

UPI-452 Accelerates iAPX 286 Bus Performance

Order Number: 292018-001

UPI-452 ACCELERATES iAPX 286 BUS PERFORMANCE

CONTENTS

PAGE

INTRODUCTION	4-4
UPI-452 iAPX 286 System Configuration	4-4
Host UPI-452 FIFO Slave Interface	4-4
UPI-452 Initialization	4-6
FIFO Data Structures	4-7
FIFO Input/Output Channel Size	4-12
Interrupt Response Timing	4-13
DMA	4-15
Host FIFO DMA	4-15
UPI-452 Internal DMA Processor	4-17
Interface Latency	4-18
FIFO DMA Freeze Mode Interface	4-19

LIST OF FIGURES, TABLES AND EQUATIONS

FIGURES:

1. iAPX 286/UPI-452 System Block Diagram	4-5
2. Input FIFO Channel Functional Diagram	4-8
3. Output FIFO Channel Functional Diagram	4-9
4. Full Duplex FIFO Operation	4-12
5. Entire FIFO Assigned to Output Channel	4-13
6. UPI-452 Command Cycle Timing	4-16
7. Disabling FIFO to Host Slave Interface Timing Diagram	4-20
8. Sequence of Events to Invoke FIFO DMA Freeze Mode	4-21
9. Sequence of Events to Invoke FIFO DMA Freeze Mode Timings	4-21

TABLES:

1. FIFO Special Function Register Default Values	4-6
2. UPI-452 Initialization Event Sequence Example	4-6
3. UPI-452 External Address Decoding ..	4-10
4. UPI-452 to Host Interrupt Sources	4-14

CONTENTS

PAGE

5. Host UPI-452 Data Transfer Performance	4-15
6. UPI-452 Internal DMA Controller Cycle Timings	4-17
7. Host DMA FIFO Data Transfer Times	4-18
8. UPI-452 Internal DMA FIFO Data Transfer Times	4-19
9. Typical Instruction Cycle Timings	4-19
10. Slave Bus Interface Status During FIFO DMA Freeze Mode	4-22
11. FIFO SFR's Characteristics During FIFO DMA Freeze Mode	4-22

DMA transfer between the Host and UPI-452 are controlled by the Host processor DMA controller. In the example shown in Figure 1, the Host DMA controller is the 8238 Advanced DMA Controller. An internal DMA transfer to or from the FIFO, as well as between other internal elements, is controlled by the UPI-452 internal DMA processor. The internal DMA processor can also transfer data between Input and Output FIFO channels directly. The description that follows details the UPI-452 interface from both the Host processor's and the UPI-452's internal CPU perspective.

One of the unique features of the UPI-452 FIFO is its ability to distinguish between commands and data embedded in the same data block. Both interrupts and status flags are provided to support this operation in either direction of data transfer. These flags and interrupts are triggered by the FIFO logic independent of and transparent to either the Host or internal CPU. Commands embedded in the data block or stream are called Data Stream Commands.

Programmable FIFO channel thresholds are another unique feature of the UPI-452. The thresholds provide for interrupting the Host only when the threshold number of bytes can be read or written to the FIFO buffer. This further decouples the Host UPI-452 interface by relieving the Host of polling the buffer to determine the number of bytes that can be read or written. It also reduces the chances of overrun and under-run errors which must be processed.

The UPI-452 also provides a means of bypassing the FIFO, in both directions, for an immediate interrupt of either the Host or internal CPU. These commands are called Immediate Commands. A complete description of the internal FIFO logic operation is given in the FIFO Data Structure section.

CONTENTS

PAGE

EQUATIONS:

1. Host Interrupt Response Time	4-14
2. Host FIFO DMA Transfer Rate—Input or Output Channel	4-15
3. Minimum Host/FIFO Transfer Rate Including Data Stream Command(s)	4-17
4. Effective Internal FIFO Transfer Time Using Internal DMA	4-19
5. Effective FIFO Transfer Time Using Individual Instructions	4-19

The UPI-452 interfaces to the IAPX 286 microprocessor as a standard Intel slave peripheral device. READ, WRITE, CS and address lines from the Host are used to access all of the Host addressable UPI-452 32-bit Position Registers (SFR).

The UPI-452 combines an MSC-31 microcontroller with 256 bytes of on-chip RAM and 8K bytes of ROM. It also contains a two channel DMA controller twice that of the MSC-31, a two channel, bidirectional and a sophisticated 128 byte, two channel, bidirectional FIFO in a single device. The UPI-452 retains all of the MSC-31 architecture, and is fully compatible with the MSC-31 instruction set.

This application note is a description of an IAPX 286 to UPI-452 slave interface. Included is a discussion of the respective timing and design considerations. This application note is meant as a supplement to the UPI-452 Advanced Data Sheet. The user should consult the data sheet for additional details on the various UPI-452 functions and features.

UPI-452 IAPX 286 SYSTEM CONFIGURATION

The interface described in this application note is shown in Figure 1. IAPX 286 UPI-452 system block diagram. The IAPX 286 system is configured in a local bus architecture design. DMA between the Host and the UPI-452 is supported by the 8238 Advanced DMA Controller. The Host microprocessor accesses all UPI-452 externally addressable registers through address decoding (see Table 1, UPI-452 External Address Decoding). The timing and interface descriptions for the UPI-452 are given in equation form with examples of specific calculations. The goal of this application note is a set of interface analysis equations. These equations are the tools a system designer can use to fully utilize the features of the UPI-452 to achieve maximum system performance.

INTRODUCTION

The UPI-452 targets the leading problem in peripheral to host interfacing, the interface of a slow peripheral with a fast Host or "bus utilization". The solution is data buffering to reduce the delay and overhead of transferring data between the Host microprocessor and I/O subsystem. The Intel CMOS UPI-452 solves this problem by combining a sophisticated programmable FIFO buffer and a slave interface with an MSC-51 based microcontroller.

The UPI-452 is Intel's newest Universal Peripheral Interface family member. The UPI-452 FIFO buffer enables Host—peripheral communications to be through streams or bursts of data rather than by individual bytes. In addition the FIFO provides a means of embedding commands within a stream or block of data. This enables the system designer to manage data and commands to further off-load the Host.

The UPI-452 interfaces to the iAPX 286 microprocessor as a standard Intel slave peripheral device. READ, WRITE, CS and address lines from the Host are used to access all of the Host addressable UPI-452 Special Function Registers (SFR).

The UPI-452 combines an MSC-51 microcontroller, with 256 bytes of on-chip RAM and 8K bytes of ROM, twice that of the 80C51, a two channel DMA controller and a sophisticated 128 byte, two channel, bidirectional FIFO in a single device. The UPI-452 retains all of the 80C51 architecture, and is fully compatible with the MSC-51 instruction set.

This application note is a description of an iAPX 286 to UPI-452 slave interface. Included is a discussion of the respective timings and design considerations. This application note is meant as a supplement to the UPI-452 Advance Data Sheet. The user should consult the data sheet for additional details on the various UPI-452 functions and features.

UPI-452 IAPX 286 SYSTEM CONFIGURATION

The interface described in this application note is shown in Figure 1, iAPX 286 UPI-452 System Block Diagram. The iAPX 286 system is configured in a local bus architecture design. DMA between the Host and the UPI-452 is supported by the 82258 Advanced DMA Controller. The Host microprocessor accesses all UPI-452 externally addressable registers through address decoding (see Table 3, UPI-452 External Address Decoding). The timings and interface descriptions below are given in equation form with examples of specific calculations. The goal of this application note is a set of interface analysis equations. These equations are the tools a system designer can use to fully utilize the features of the UPI-452 to achieve maximum system performance.

HOST-UPI-452 FIFO SLAVE INTERFACE

The UPI-452 FIFO acts as a buffer between the external Host 80286 and the internal CPU. The FIFO allows the Host - peripheral interface to achieve maximum decoupling of the interface. Each of the two FIFO channels is fully user programmable. The FIFO buffer ensures that the respective CPU, Host or internal CPU, receives data in the same order as transmitted. Three slave bus interface handshake methods are supported by the UPI-452; DMA, Interrupt and Polled.

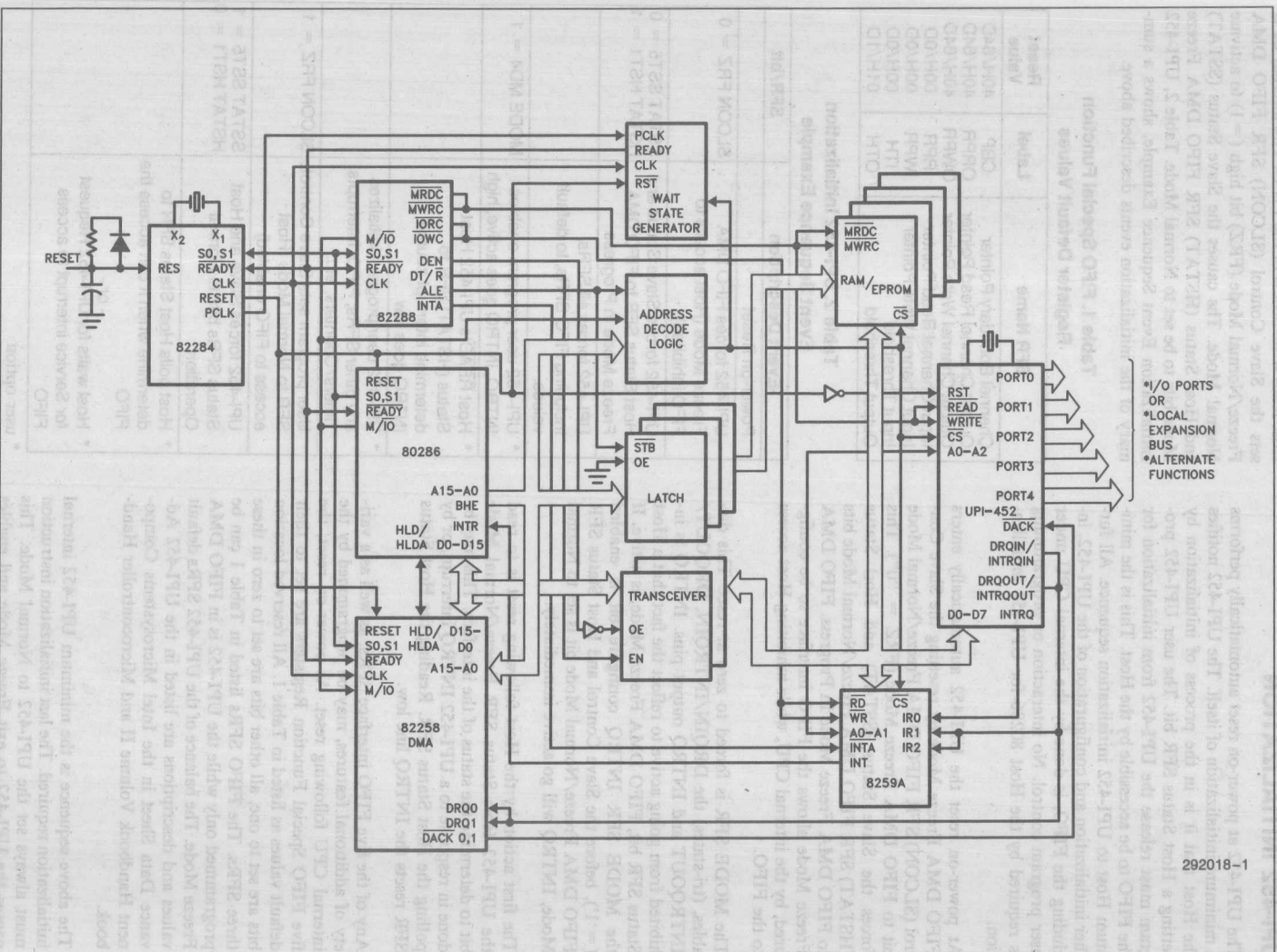
The interface between the Host 80286 and the UPI-452 is accomplished with a minimum of signals. The 8 bit data bus plus READ, WRITE, CS, and A0-2 provide access to all of the externally addressable UPI-452 registers including the two FIFO channels. Interrupt and DMA handshaking pins are tied directly to the interrupt controller and DMA controller respectively.

DMA transfers between the Host and UPI-452 are controlled by the Host processors DMA controller. In the example shown in Figure 1, the Host DMA controller is the 82258 Advanced DMA Controller. An internal DMA transfer to or from the FIFO, as well as between other internal elements, is controlled by the UPI-452 internal DMA processor. The internal DMA processor can also transfer data between Input and Output FIFO channels directly. The description that follows details the UPI-452 interface from both the Host processor's and the UPI-452's internal CPU perspective.

One of the unique features of the UPI-452 FIFO is its ability to distinguish between commands and data embedded in the same data block. Both interrupts and status flags are provided to support this operation in either direction of data transfer. These flags and interrupts are triggered by the FIFO logic independent of, and transparent to either the Host or internal CPUs. Commands embedded in the data block, or stream, are called Data Stream Commands.

Programmable FIFO channel Thresholds are another unique feature of the UPI-452. The Thresholds provide for interrupting the Host only when the Threshold number of bytes can be read or written to the FIFO buffer. This further decouples the Host UPI-452 interface by relieving the Host of polling the buffer to determine the number of bytes that can be read or written. It also reduces the chances of overrun and underrun errors which must be processed.

The UPI-452 also provides a means of bypassing the FIFO, in both directions, for an immediate interrupt of either the Host or internal CPU. These commands are called Immediate Commands. A complete description of the internal FIFO logic operation is given in the FIFO Data Structure section.



292018-1

UPI-452 INITIALIZATION

The UPI-452 at power-on reset automatically performs a minimum initialization of itself. The UPI-452 notifies the Host that it is in the process of initialization by setting a Host Status SFR bit. The user UPI-452 program must release the UPI-452 from initialization for the FIFO to be accessible by the Host. This is the minimum Host to UPI-452 initialization sequence. All further initialization and configuration of the UPI-452, including the FIFO, is done by the internal CPU under user program control. No interaction or programming is required by the Host 80286 for UPI-452 initialization.

At power-on reset the UPI-452 automatically enters FIFO DMA Freeze Mode by resetting the Slave Control (SLCON) SFR FIFO DMA Freeze/Normal Mode bit to FIFO DMA Freeze Mode (FRZ = "0"). This forces the Slave Status (SSTAT) and Host Status (HSTAT) SFR FIFO DMA Freeze/Normal Mode bits to FIFO DMA Freeze Mode In Progress. FIFO DMA Freeze Mode allows the FIFO interface to be configured, by the internal CPU, while inhibiting Host access to the FIFO.

The MODE SFR is forced to zero at reset. This disables, (tri-states) the DRQIN/INTRQIN, DRQOUT/INTRQOUT and INTRQ output pins. INTRQ is inhibited from going active to reflect the fact that a Host Status SFR bit, FIFO DMA Freeze Mode, is active. If the MODE SFR INTRQ configure bit is enabled (= '1'), before the Slave Control and Host Status SFR FIFO DMA Freeze/Normal Mode bit is set to Normal Mode, INTRQ will go active immediately.

The first action by the Host following reset is to read the UPI-452 Host Status SFR Freeze/Normal Mode bit to determine the status of the interface. This may be done in response to a UPI-452 INTRQ interrupt, or by polling the Host Status SFR. Reading the Host Status SFR resets the INTRQ line low.

Any of the five FIFO interface SFRs, as well as a variety of additional features, may be programmed by the internal CPU following reset. At power-on reset, the five FIFO Special Function Registers are set to their default values as listed in Table 1. All reserved location bits are set to one, all other bits are set to zero in these three SFRs. The FIFO SFRs listed in Table 1 can be programmed only while the UPI-452 is in FIFO DMA Freeze Mode. The balance of the UPI-452 SFRs default values and descriptions are listed in the UPI-452 Advance Data Sheet in the Intel Microsystems Component Handbook Volume II and Microcontroller Handbook.

The above sequence is the minimum UPI-452 internal initialization required. The last initialization instruction must always set the UPI-452 to Normal Mode. This causes the UPI-452 to exit Freeze Mode and enables

Host read/write access of the FIFO. The internal CPU sets the Slave Control (SLCON) SFR FIFO DMA Freeze/Normal Mode (FRZ) bit high (= 1) to activate Normal Mode. This causes the Slave Status (SSTAT) and Host Status (HSTAT) SFR FIFO DMA Freeze Mode bits to be set to Normal Mode. Table 2, UPI-452 Initialization Event Sequence Example, shows a summary of the initialization events described above.

Table 1. FIFO Special Function Register Default Values

SFR Name	Label	Reset Value
Channel Boundary Pointer	CBP	40H/64D
Output Channel Read Pointer	ORPR	40H/64D
Output Channel Write Pointer	OWPR	40H/64D
Input Channel Read Pointer	IRPR	00H/0D
Input Channel Write Pointer	IWPR	00H/0D
Input Threshold	ITH	00H/0D
Output Threshold	OTH	01H/1D

Table 2. UPI-452 Initialization Event Sequence Example

Event Description	SFR/bit
Power-on Reset	
UPI-452 forces FIFO DMA Freeze Mode (Host access to FIFO inhibited)	SLCON FRZ = 0
UPI-452 forces Slave Status and Host Status SFR to FIFO DMA Freeze Mode In Progress	SSTAT SST5 = 0 HSTAT HST1 = 1
UPI-452 forces all SFRs, including FIFO SFRs, to default values.	
* UPI-452 user program enables INTRQ, INTRQ goes active, high	MODE MD4 = 1
* Host READ's UPI-452 Host Status (HSTAT) SFR to determine interrupt source, INTRQ goes low	
* UPI-452 user program initializes any other SFRs; FIFO, Interrupts, Timers/Counters, etc.	
User program sets Slave Control SFR to Normal Mode (Host access to FIFO enabled)	SLCON FRZ = 1
UPI-452 forces Slave and Host Status SFRs bits to Normal Operation	SSTAT SST5 = 1 HSTAT HST1 = 0
* Host polls Host Status SFR to determine when it can access the FIFO	
- or -	
* Host waits for UPI-452 Request for Service interrupt to access FIFO	

* user option

FIFO DATA STRUCTURES

Overview

The UPI-452 provides three means of communication between the Host microprocessor and the UPI-452 in either direction;

- Data
- Data Stream Commands
- Immediate Commands

Data and Data Stream Commands (DSC) are transferred between the Host and UPI-452 through the UPI-452 FIFO buffer. The third, Immediate Commands, provides a means of bypassing the FIFO entirely. These three data types are in addition to direct access by either Host or Internal CPU of dedicated Status and Control Special Function Registers (SFR).

The FIFO appears to both the Host 80286 and the internal CPU as 8 bits wide. Internally the FIFO is logically nine bits wide. The ninth bit indicates whether the byte is a data or a Data Stream Command (DSC) byte; 0 = data, 1 = DSC. The ninth bit is set by the FIFO logic in response to the address specified when writing to the FIFO by either Host or internal CPU. The FIFO uses the ninth bit to condition the UPI-452 interrupts and status flags as a byte is made available for a Host or internal CPU read from the FIFO. Figures 2 and 3 show the structure of each FIFO channel and the logical ninth bit.

It is important to note that both data and DSCs are actually entered into the FIFO buffer (see Figures 2 and 3). External addressing of the FIFO determines the state of the internal FIFO logic ninth bit. Table 3 shows the UPI-452 External Address Decoding used by the Host and the corresponding action. The internal CPU interface to the FIFO is essentially identical to the external Host interface. Dedicated internal Special Function Registers provide the interface between the FIFO, internal CPU and the internal two channel DMA processor. FIFO read and write operations by the Host and internal CPU are interleaved by the UPI-452 so they appear to be occurring simultaneously.

The ninth bit provides a means of supporting two data types within the FIFO buffer. This feature enables the Host and UPI-452 to transfer both commands and data while maintaining the decoupled interface a FIFO buffer provides. The logical ninth bit provides both a means of embedding commands within a block of data and a means for the internal CPU, or external Host, to discriminate between data and commands. Data or DSCs may be written in any order desired. Data Stream

Commands can be used to structure or dispatch the data by defining the start and end of data blocks or packets, or how the data following a DSC is to be processed.

A Data Stream Command (DSC) acts as an internal service routine vector. The DSC generates an interrupt to a service routine which reads the DSC. The DSC byte acts as an address vector to a user defined service routine. The address can be any program or data memory location with no restriction on the number of DSCs or address boundaries.

A Data Stream Command (DSC) can also be used to clear data from the FIFO or "FLUSH" the FIFO. This is done by appending a DSC to the end of a block of data entered in the FIFO which is less than the programmed threshold number of bytes. The DSC will cause an interrupt, if enabled, to the respective receiving CPU. This ensures that a less than Threshold number of bytes in the FIFO will be read. Two conditions force a Request for Service interrupt, if enabled, to the Host. The first is due to a Threshold number of bytes having been written to the FIFO Output channel; the second is if a DSC is written to the Output FIFO channel. If less than the Threshold number of bytes are written to the Output FIFO channel, the Host Status SFR flag will not be set, and a Request for Service interrupt will not be generated, if enabled. By appending a DSC to end of the data block, the FIFO Request for Service flag and/or interrupt will be generated.

An example of a FIFO Flush application is a mass storage subsystem. The UPI-452 provides the system interface to a subsystem which supports tape and disk storage. The FIFO size is dynamically changed to provide the maximum buffer size for the direction of transfer. Large data blocks are the norm in this application. The FIFO Flush provides a means of purging the FIFO of the last bytes of a transfer. This guarantees that the block, no matter what its size, will be transmitted out of the FIFO.

Immediate Commands allow more direct communication between the Host processor and the UPI-452 by bypassing the FIFO in either direction. The Immediate Command IN and OUT SFRs are two more unique address locations externally and internally addressable. Both DSCs and Immediate Commands have internal interrupts and interrupt priorities associated with their operation. The interrupts are enabled or disabled by setting corresponding bits in the Slave Control (SLCON), Interrupt Enable (IEC), Interrupt Priority (IPC) and Interrupt Enable and Priority (IEP) SFRs. A detailed description of each of these may be found in the UPI-452 Advance Information Data Sheet.

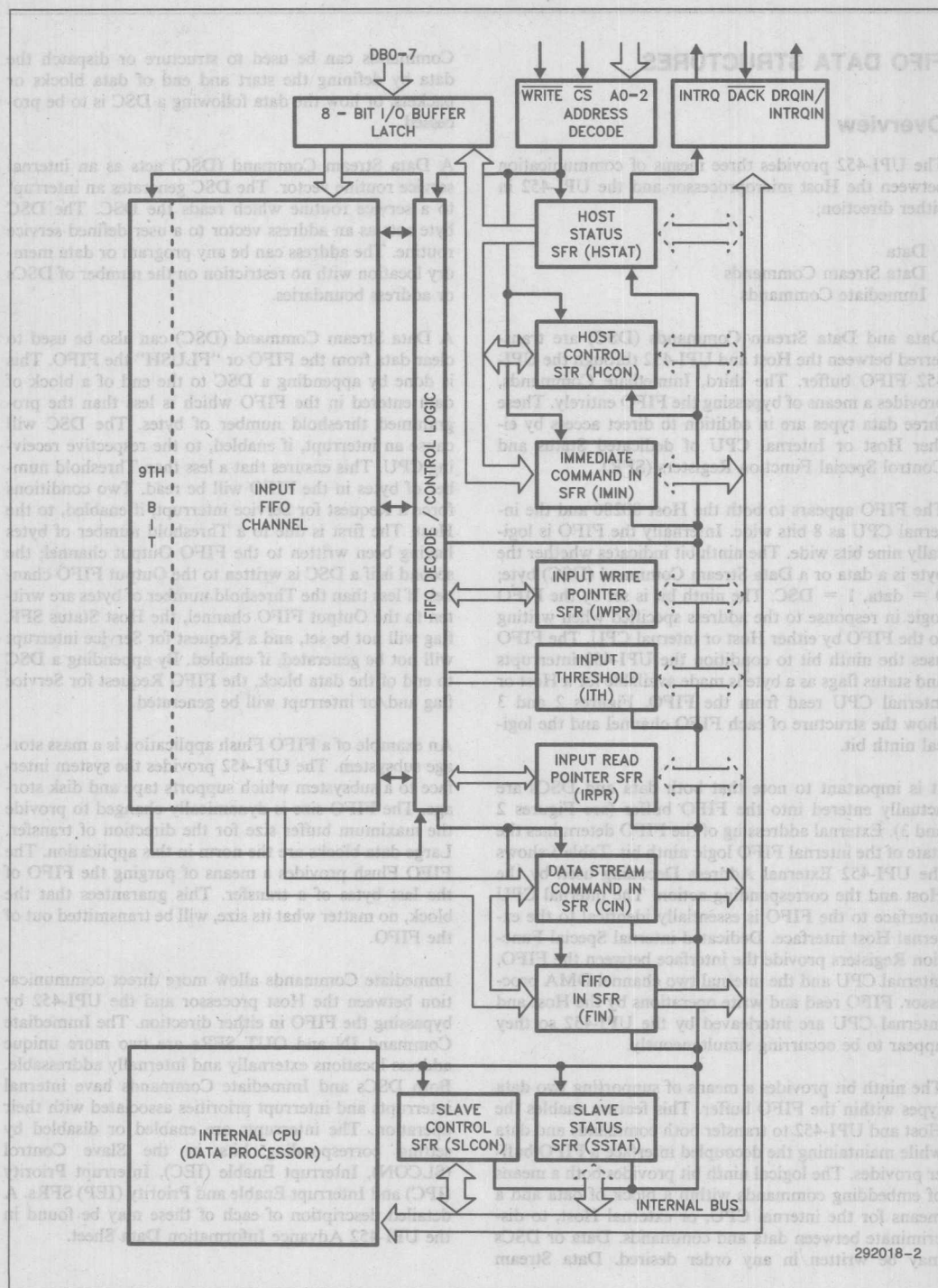


Figure 2. Input FIFO Channel Functional Diagram

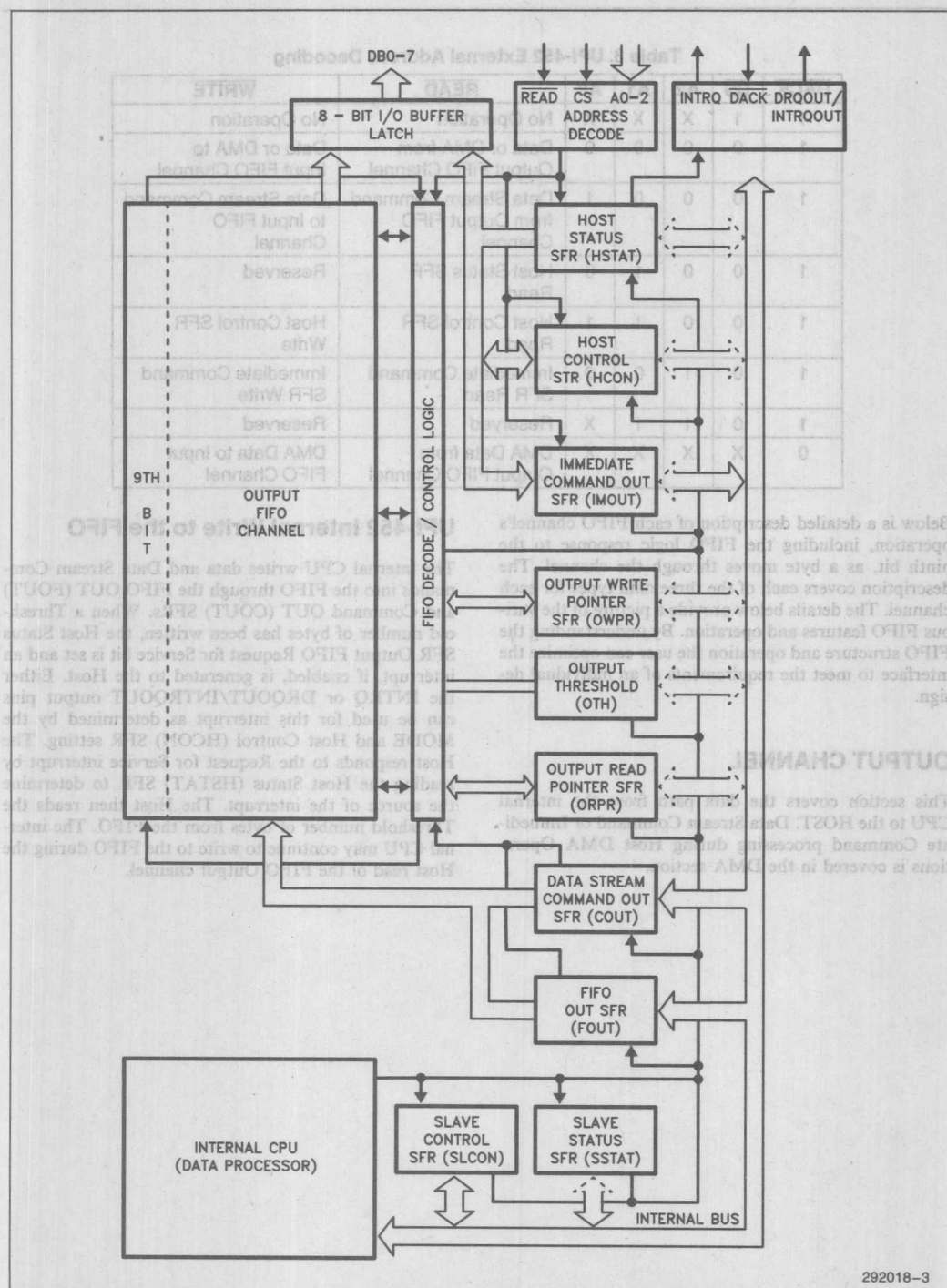


Figure 3. Output FIFO Channel Functional Diagram

READ					WRITE
1	1	X	X	X	No Operation
1	0	0	0	0	Data or DMA from Output FIFO Channel
1	0	0	0	1	Data Stream Command from Output FIFO Channel
1	0	0	1	0	Host Status SFR Read
1	0	0	1	1	Host Control SFR Read
1	0	1	0	0	Immediate Command SFR Read
1	0	1	1	X	Reserved
0	X	X	X	X	DMA Data from Output FIFO Channel

Below is a detailed description of each FIFO channel's operation, including the FIFO logic response to the ninth bit, as a byte moves through the channel. The description covers each of the three data types for each channel. The details below provide a picture of the various FIFO features and operation. By understanding the FIFO structure and operation the user can optimize the interface to meet the requirements of an individual design.

OUTPUT CHANNEL

This section covers the data path from the internal CPU to the HOST. Data Stream Command or Immediate Command processing during Host DMA Operations is covered in the DMA section.

UPI-452 Internal Write to the FIFO

The internal CPU writes data and Data Stream Commands into the FIFO through the FIFO OUT (FOUT) and Command OUT (COUT) SFRs. When a Threshold number of bytes has been written, the Host Status SFR Output FIFO Request for Service bit is set and an interrupt, if enabled, is generated to the Host. Either the INTRQ or DRQOUT/INTRQOUT output pins can be used for this interrupt as determined by the MODE and Host Control (HCON) SFR setting. The Host responds to the Request for Service interrupt by reading the Host Status (HSTAT) SFR to determine the source of the interrupt. The Host then reads the Threshold number of bytes from the FIFO. The internal CPU may continue to write to the FIFO during the Host read of the FIFO Output channel.

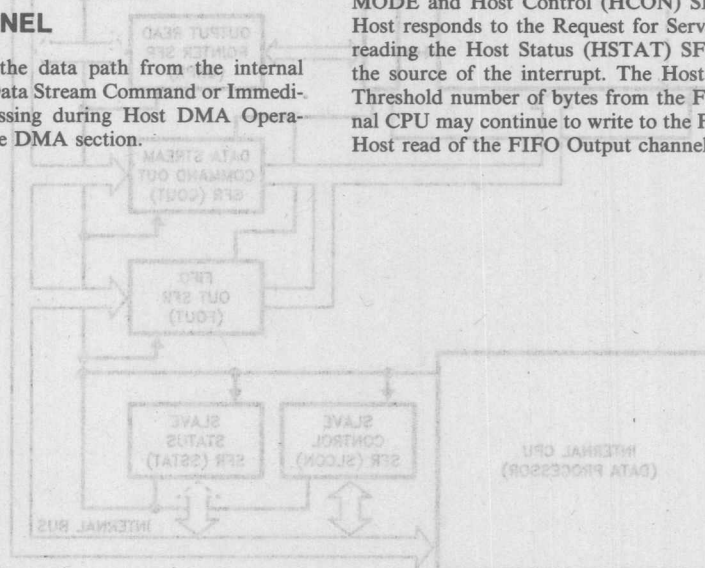


Figure 3. Output FIFO Channel Functional Diagram

Data Stream Commands may be written to the Output FIFO channel at any time during a write of data bytes. The write instruction need only specify the Command Out (COUT) SFR in the direct register instruction used. Immediate Commands may also be written at any time to the Immediate Command OUT (IMOUT) SFR. The Host reads Immediate Commands from the Immediate Command OUT (IMOUT).

The internal CPU can determine the number of bytes to write to the FIFO Output channel in one of three ways. The first, and most efficient, is by utilizing the internal DMA processor which will automatically manage the writing of data to avoid Underrun or Overrun Errors. The second is for the internal CPU to read the Output FIFO channels Read and Write Pointers and compare their values to determine the available space. The third method for determining the available FIFO space is to always write the programmed channel size number of bytes to the Output FIFO. This method would use the Overrun Error flag and interrupt to halt FIFO writing whenever the available space was less than the channel size. The interrupt service routine could read the channel pointers to determine or monitor the available channel space. The time required for the internal CPU to write data to the Output FIFO channel is a function of the individual instruction cycle time and the number of bytes to be written.

Host Read from the FIFO

The Host reads data or Data Stream Commands (DSC) from the FIFO in response to the Host Status (HSTAT) SFR flags and interrupts, if enabled. All Host read operations access the same UPI-452 internal I/O Buffer Latch. At the end of the previous Host FIFO read cycle a byte is loaded from the FIFO into the I/O Buffer Latch and Host Status (HSTAT) SFR bit 5 is set or cleared (1 = DSC, 0 = data) to reflect the state of the byte's FIFO ninth bit. If the FIFO ninth bit is set (= 1) indicating a DSC, an interrupt is generated to the external Host via INTRQ pin or INTRQIN/INTRQOUT pins as determined by Host Control (HCON) SFR bit 1. The Host then reads the Host Status (HSTAT) SFR to determine the source of the interrupt.

The most efficient Host read operation of the FIFO Output channel is through the use of Host DMA. The UPI-452 fully supports external DMA handshaking. The MODE and Host Control SFRs control the configuration of UPI-452 Host DMA handshake outputs. If Host DMA is used the Threshold Request for Service interrupt asserts the UPI-452 DMA Request (DRQOUT) output. The Host DMA processor acknowledges with DACK which acts as a chip select of the FIFO channels. The DMA transfer would stop when either the threshold byte count had been read, as programmed in the Host DMA processor, or when the DRQOUT output is brought inactive by the UPI-452.

INPUT CHANNEL

This section covers the data path from the HOST to the internal CPU or internal DMA processor. The details of Data Stream Command or Immediate Command processing during internal DMA operations are covered in the DMA section below.

Host Write to the FIFO

The Host writes data and Data Stream Commands into the FIFO through the FIFO IN (FIN) and Command IN (CIN) SFRs. When a Threshold number of bytes has been read out of the Input FIFO channel by the internal CPU, the Host Status SFR Input FIFO Request for Service bit is set and an interrupt, if enabled, is generated to the Host. The Input FIFO Threshold interrupt tells the Host that it may write the next block of data into the FIFO. Either the INTRQ or DRQIN/INTRQIN output pins can be used for this interrupt as determined by the MODE and Host Control (HCON) SFR settings. The Host may continue to write to the FIFO Input channel during the internal CPU read of the FIFO. Data Stream Commands may be written to the FIFO Input channel at any time during a write of data bytes. Immediate Commands may also be written at any time to the Immediate Command IN (IMIN) SFR.

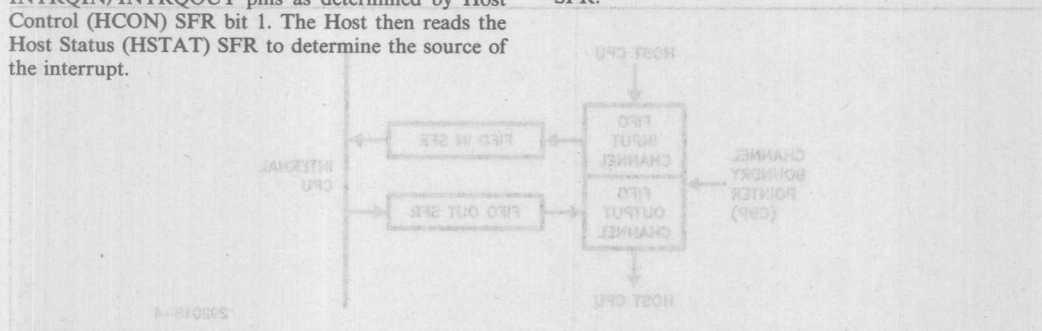


Figure 4: Full Duplex FIFO Operation

The Host also has three methods for determining the available FIFO space. Two are essentially identical to that of the internal CPU. They involve reading the FIFO Input channel pointers and using the Host Status SFR Underrun and Overrun Error flags and Request for Service interrupts these would generate, if enabled in combination. The third involves using the UPI-452 Host DMA controller handshake signals and the programmed Input FIFO Threshold. The Host would receive a Request for Service interrupt when an Input FIFO channel has a Threshold number of bytes able to be written by the Host. The Host service routine would then write the Threshold number of bytes to the FIFO.

If a Host DMA is used to write to the FIFO Input channel, the Threshold Request for Service interrupt could assert the UPI-452 DRQIN output. The Host DMA processor would assert DACK and the FIFO write would be completed by Host the DMA processor. The DMA transfer would stop when either the Threshold byte count had been written or the DRQIN output was removed by the UPI-452. Additional details on Host and internal DMA operation is given below.

Internal Read of the FIFO

At the end of an internal CPU read cycle a byte is loaded from the FIFO buffer into the FIFO IN/Command IN SFR and Slave Status (SSTAT) SFR bit 1 is set or cleared (1 = data, 0 = DSC) to reflect the state of the FIFO ninth bit. If the byte is a DSC, the FIFO ninth bit is set (= 1) and an interrupt is generated, if enabled, to the Internal CPU. The internal CPU then reads the Slave Status (SSTAT) SFR to determine the source of the interrupt.

Immediate Commands are written by the Host and read by the internal CPU through the Immediate Command IN (IMIN) SFR. Once written, an Immediate Command sets the Slave Status (SSTAT) SFR flag bit and generates an interrupt, if enabled, to the internal CPU. In response to the interrupt the internal CPU

reads the Slave Status (SSTAT) SFR to determine the source of the interrupt and service the Immediate Command.

FIFO INPUT/OUTPUT CHANNEL SIZE

Host

The Host does not have direct control of the FIFO Input or Output channel sizes or configuration. The Host can, however, issue Data Stream Commands or Immediate Commands to the UPI-452 instructing the UPI-452 to reconfigure the FIFO interface by invoking FIFO DMA Freeze Mode. The Data Stream Command or Immediate Command would be a vector to a service routine which performs the specific reconfiguration.

UPI-452 Internal

The default power-on reset FIFO channel sizes are listed in the "Initialization" section and can be set only by the internal CPU during FIFO DMA Freeze Mode. The FIFO channel size is selected to achieve the optimum application performance. The entire 128 byte FIFO can be allocated to either the Input or Output channel. In this case the other channel consists of a single SFR; FIFO IN/Command IN or FIFO OUT/Command OUT SFR. Figure 4 shows a FIFO division with a portion devoted to each channel. Figure 5 shows a FIFO configuration with all 128 bytes assigned to the Output channel.

The FIFO channel Threshold feature allows the user to match the FIFO channel size and the performance of the internal and Host data transfer rates. The programmed Threshold provides an elasticity to the data transfer operation. An example is if the Host FIFO

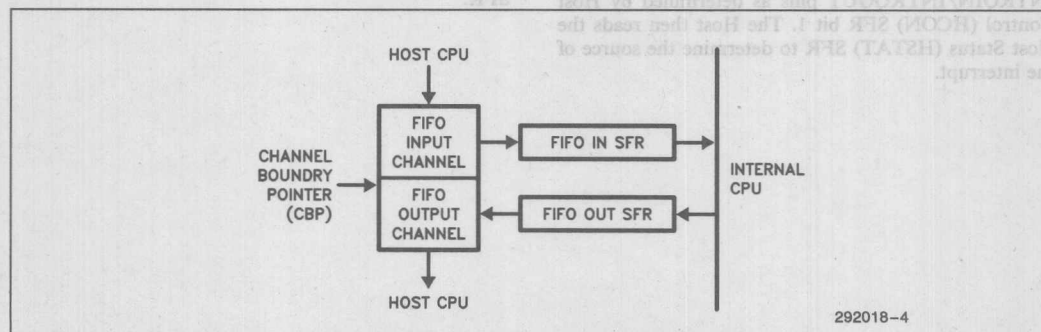


Figure 4. Full Duplex FIFO Operation

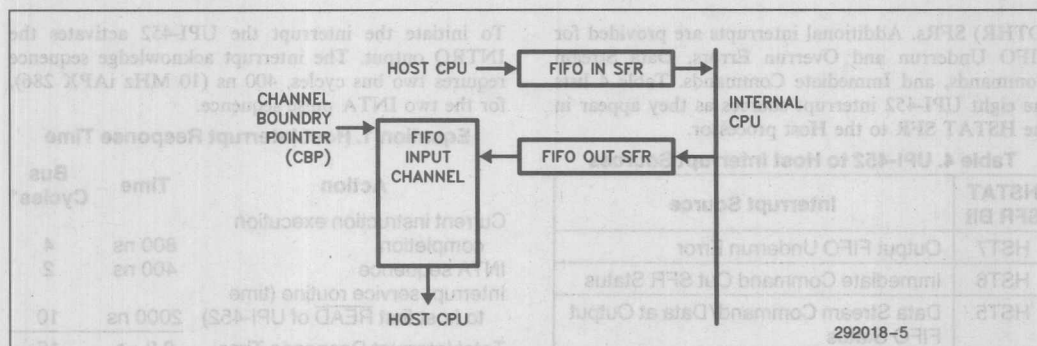


Figure 5. Entire FIFO Assigned to Output Channel

data transfer rate is twice as fast as the internal FIFO DMA data transfer rate. In this example the FIFO Input channel size is programmed to be 64 bytes and the Input channel Threshold is programmed to be 20 bytes. The Host writes the first 64 bytes to the Input FIFO. When the internal DMA processor has read 20 bytes the Threshold interrupt, or DMA request (DRQIN), is generated to signal the Host to begin writing more data to the Input FIFO channel. The internal DMA processor continues to read data from the Input channel as the Host, or Host DMA processor, writes to the FIFO. The Host can write 40 bytes to the FIFO Input channels in the time it takes for the internal DMA processor to read 20 more bytes from it. This will keep both the Host and internal DMA operating at their maximum rates without forcing one to wait for the other.

Two methods of managing the FIFO size are possible; fixed and variable channel size. A fixed channel size is one where the channel is configured at initialization and remains unchanged throughout program execution. In a variable FIFO channel size, the configuration is changed dynamically to meet the data transmission requirements as needed. An example of a variable channel size application is the mass storage subsystem described earlier. To meet the demands of a large data block transfer the FIFO size could be fully allocated to the Input or Output channel as needed. The Thresholds are also reprogrammed to match the respective data transfer rates.

An example of a fixed channel size application might be one which uses the UPI-452 to directly control a series of stepper motors. The UPI-452 manages the motor operation and status as required. This would include pulse train, acceleration, deceleration and feedback. The Host transmits motor commands to the UPI-452 in blocks of 6–10 bytes. Each block of motor command data is preceded by a command to the UPI-452 which selects a specific motor. The UPI-452 transmits blocks of data to the Host which provides motor and overall system status. The data and embedded commands structure, indicating the specific motor, is the same. In

this example the default 64 bytes per channel might be adequate for both channels.

INTERRUPT RESPONSE TIMING

Interrupts enable the Host UPI-452 FIFO buffer interface and the internal CPU FIFO buffer interface to operate with a minimum of overhead on the respective CPU. Each CPU is “interrupted” to service the FIFO on an as needed basis only. In configuring the FIFO buffer Thresholds and choosing the appropriate internal DMA Mode the user must take into account the interrupt response time for both CPUs. These response times will affect the DMA transfer rates for each channel. By choosing FIFO channel Thresholds which reflect both the respective DMA transfer rate and the interrupt response time the user will achieve the maximum data throughput and system bus decoupling. This in turn will mean the overall available system bus bandwidth will increase.

The following section describes the FIFO interrupt interface to the Host and internal CPU. It also describes an analysis of sample interrupt response times for the Host and UPI-452 internal CPU. These equations and the example times shown are then used in the DMA section to further analyze an optimum Host UPI-452 interface.

HOST

Interrupts to the Host processor are supported by the three UPI-452 output pins; INTRQ, DRQIN/INTRQIN and DRQOUT/INTRQOUT. INTRQ is a general purpose Request For Service interrupt output. DRQIN/INTRQIN and DRQOUT/INTRQOUT pins are multiplexed to provide two special “Request for Service” FIFO interrupt request lines when DMA is disabled. A FIFO Input or Output channel Request for Service interrupt is generated based upon the value programmed in the respective channel’s Threshold SFRs; Input Threshold (ITHR), and Output Threshold

(OTHR) SFRs. Additional interrupts are provided for FIFO Underrun and Overrun Errors, Data Stream Commands, and Immediate Commands. Table 4 lists the eight UPI-452 interrupt sources as they appear in the HSTAT SFR to the Host processor.

Table 4. UPI-452 to Host Interrupt Sources

HSTAT SFR Bit	Interrupt Source
HST7	Output FIFO Underrun Error
HST6	Immediate Command Out SFR Status
HST5	Data Stream Command/Data at Output FIFO Status
HST4	Output FIFO Request for Service Status
HST3	Input FIFO Overrun Error Condition
HST2	Immediate Command In SFR Status
HST1	FIFO DMA Freeze/Normal Mode Status
HST0	Input FIFO Request for Service

The interrupt response time required by the Host processor is application and system specific. In general, a typical sequence of Host interrupt response events and the approximate times associated with each are listed in Equation 1.

The example assumes the hardware configuration shown in Figure 1, iAPX 286/UPI-452 Block Diagram, with an 8259A Programmable Interrupt Controller. The timing analysis in Equation 1 also assumes the following; no other interrupt is either in process or pending, nor is the 286 in a LOCK condition. The current instruction completion time is 8 clock cycles (800 ns @ 10 MHz), or 4 bus cycles. The interrupt service routine first executes a PUSHA instruction, PUSH All General Cycles (rounded to complete bus cycle). The next service routine instruction reads the UPI-452 Host Status SFR to determine the interrupt source.

It is important to note that any UPI-452 INTRQ interrupt service routine should ALWAYS mask for the Freeze Mode bit first. This will insure that Freeze Mode always has the highest priority. This will also save the time required to mask for bits which are forced inactive during Freeze Mode, before checking the Freeze Mode bit. Access to the FIFO channels by the Host is inhibited during Freeze Mode. Freeze Mode is covered in more detail below.

To initiate the interrupt the UPI-452 activates the INTRQ output. The interrupt acknowledge sequence requires two bus cycles, 400 ns (10 MHz iAPX 286), for the two INTA pulse sequence.

Equation 1. Host Interrupt Response Time

Action	Time	Bus Cycles*
Current instruction execution completion	800 ns	4
INTA sequence	400 ns	2
Interrupt service routine (time to host first READ of UPI-452)	2000 ns	10
Total Interrupt Response Time	2.3 μ s	16

NOTE:

10 MHz iAPX 286 bus cycle, 200 ns each

UPI-452 Internal

The internal CPU FIFO interrupt interface is essentially identical to that of the Host to the FIFO. Three internal interrupt sources support the FIFO operation; FIFO-Slave bus Interface Buffer, DMA Channel 0 and DMA Channel 1 Requests. These interrupts provide a maximum decoupling of the FIFO buffer and the internal CPU. The four different internal DMA Modes available add flexibility to the interface.

The FIFO-Slave Bus Interface interrupt response is also similar to the Host response to an INTRQ Request for Service interrupt. The internal CPU responds to the interrupt by reading the Slave Status (SSTAT) SFR to determine the source of the interrupt. This allows the user to prioritize the Slave Status flag response to meet the users application needs. The interrupt is enabled, and the interrupt priority. In general, to finish execution of the current instruction, respond to the interrupt request, push the Program Counter (PC) and vector to the first instruction of the interrupt service routine requires from 38 to 86 oscillator periods (2.38 to 5.38 μ s @ 16 MHz). If the interrupt is due to an Immediate Command or DSC, additional time is required to read the Immediate Command or DSC SFR and vector to the appropriate service routine. This means two service routines back to back. One service routine to read the Slave Status (SSTAT) SFR to determine the source of the Request for Service interrupt, and second the service routine pointed to by the Immediate Command or DSC byte read from the respective SFR.

DMA

DMA is the fastest and most efficient way for the Host or internal CPU to communicate with the FIFO buffer. The UPI-452 provides support for both of these DMA paths. The two DMA paths and operations are fully independent of each other and can function simultaneously. While the Host DMA processor is performing a DMA transfer to or from the FIFO, the UPI-452 internal DMA processor can be doing the same.

Below are descriptions of both the Host and internal DMA operations. Both DMA paths can operate asynchronously and at different transfer rates. In order to make the most of this simultaneous asynchronous operation it is necessary to calculate the two transfer rates and accurately match their operations. Matching the different transfer rates is done by a combination of accurately programmed FIFO channel size and channel Thresholds. This provides the maximum Host and internal CPU to FIFO buffer interface decoupling. Below is a description of each of the two DMA operations and sample calculations for determining transfer rates. The next section of this application note, "Interface Latency", details the considerations involved in analyzing effective transfer rates when the overhead associated with each transfer is considered.

HOST FIFO DMA

DMA transfers between the Host and UPI-452 FIFO buffer are controlled by the Host CPU's DMA controller, and is independent of the UPI-452's internal two channel DMA processor. The UPI-452's internal DMA processor supports data transfers between the UPI-452 internal RAM, external RAM (via the Local Expansion Bus) and the various Special Function Registers including the FIFO Input and Output channel SFRs.

The maximum DMA transfer rate is achieved by the minimum DMA transfer cycle time to accomplish a source to destination move. The minimum Host UPI-452 FIFO DMA cycle time possible is determined by the READ and WRITE pulse widths, UPI-452 command recovery times in relation to the DMA transfer timing and DMA controller transfer mode used. Table 5 shows the relationship between the iAPX-286, iAPX-186 and UPI-452 for various DMA as well as non-DMA byte by byte transfer modes versus processor frequencies.

Host processor speed vs wait states required with UPI-452 running at 16 MHz:

Table 5. Host UPI-452 Data Transfer Performance

Processor & Speed	Wait States: Back to Back READ/WRITE's	DMA: Single Cycle	Two Cycle
iAPX-186* 8 MHz	0	N/A*	0
10 MHz	0	N/A*	0
12.5 MHz	1	N/A*	0
iAPX-286** 6 MHz	0	0	0
8 MHz	1	1	0
10 MHz	2	2	0

NOTES:

* iAPX 186 On-chip DMA processor is two cycle operation only.

** iAPX 286 assumes 82258 ADMA (or other DMA) running 286 bus cycles at 286 clock rate.

In this application note system example, shown in Figure 1, DMA operation is assumed to be two bus cycle I/O to memory or memory to I/O. Two cycle DMA consists of a fetch bus cycle from the source and a store bus cycle to the destination. The data is stored in the DMA controller's registers before being sent to the destination. Single cycle DMA transfers involve a simultaneous fetch from the source and store to the destination. As the most common method of I/O-memory DMA operation, two cycle DMA transfers are the focus of this application note analysis. Equation 2 illustrates a calculation of the overall transfer rate between the FIFO buffer and external Host for a maximum FIFO size transfer. The equation does not account for the latency of initiating the DMA transfer.

Equation 2. Host FIFO DMA Transfer Rate—Input or Output Channel

$$\begin{aligned}
 &2 \text{ Cycle DMA Transfer-I/O (UPI-452) to System Memory} \\
 &= \text{FIFO channel size} * (\text{DMA READ/WRITE FIFO time} + \text{DMA WRITE/READ Memory Time}) \\
 &= 128 \text{ bytes} * (200 \text{ ns} + 200 \text{ ns}) \\
 &= 51.2 \mu\text{s} \\
 &= 256 \text{ bus cycles} *
 \end{aligned}$$

NOTES:

*10 MHz iAPX 286, 200 ns bus cycles.

The UPI-452 design is optimized for high speed DMA transfers between the Host and the FIFO buffer. The

UPI-452 internal FIFO buffer control logic provides the necessary synchronization of the external Host event and the internal CPU machine cycle during UPI-452 RD/WR accesses. This internal synchronization is addressed by the TCC AC specification of the UPI-452 shown in Figure 6. TCC is the time from the leading or trailing edge of a UPI-452 RD/WR to the same edge of the next UPI-452 RD/WR. The TCC time is effectively another way of measuring the system bus cycle time with reference to UPI-452 accesses.

In the iAPX-286 10 MHz system depicted in this application note the bus cycle time is 200 ns. Alternate cycle accesses of the UPI-452 during two cycle DMA operation yields a TCC time of 400 ns which is more than the TCC minimum time of 375 ns. Back to back Host UPI-452 READ/WRITE accesses may require wait states as shown in Table 5. The difference between 10 MHz iAPX-186 and 10 MHz iAPX 286 required wait states is due to the number of clock cycles in the respective bus cycle timings. The four clocks in a 10 MHz iAPX 186 bus cycle means a minimum TCC time of 400 ns versus 200 ns for a 10 MHz iAPX 286 with two clock cycle zero wait state bus cycle.

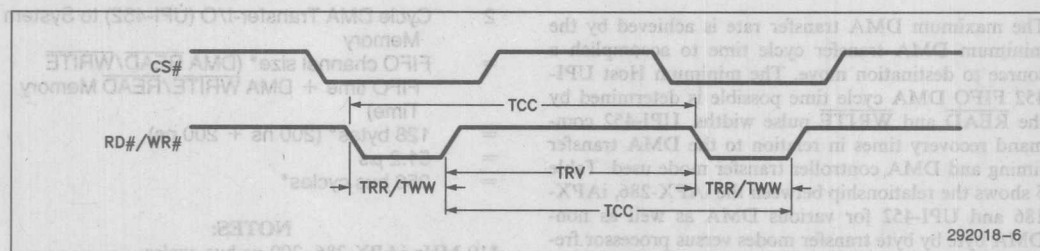
DMA handshaking between the Host DMA controller and the UPI-452 is supported by three pins on the UPI-452: DRQIN/INTRQIN, DRQOUT/INTRQOUT and DACK. The DRQIN/INTRQIN and DRQOUT/INTRQOUT outputs are two multiplexed DMA or interrupt request pins. The function of these pins is controlled by MODE SFR bit 6 (MD6). DRQIN and DRQOUT provide a direct interface to the Host system DMA controller (see Figure 1). In response to a DRQIN or DRQOUT request, the Host DMA controller initiates control of the system bus using HLD/HLDA. The FIFO Input or Output channel transfer is accomplished with a minimum of Host overhead and system bus bandwidth.

The third handshake signal pin is $\overline{\text{DACK}}$ which is used as a chip select during DMA data transfers. The UPI-452 Host READ and WRITE input signals select the respective Input and Output FIFO channel during DMA transfers. The $\overline{\text{CS}}$ and address lines provide DMA acknowledge for processors with onboard DMA controllers which do not generate a $\overline{\text{DACK}}$ signal.

The iAPX 286 Block I/O Instructions provide an alternative to two cycle DMA data transfers with approximately the same data rate. The String Input and Output instructions (INS & OUTS) when combined with the Repeat (REP) prefix, modifies INS and OUTS to provide a means of transferring blocks of data between I/O and Memory. The data transfer rate using REP INS/OUTS instructions is calculated in the same way as two cycle DMA transfer times. Each READ or WRITE would be 200 ns in a 10 MHz iAPX 286 system. The maximum transfer rate possible is 2.5 MBytes/second. The Block I/O FIFO data transfer calculation is the same as that shown in Equation 2 for two cycle DMA data transfers including TCC timing effects.

FIFO Data Structure and Host DMA

During a Host DMA write to the FIFO, if a DSC is to be written, the DMA transfer is stopped, the DSC is written and the DMA restarted. During a Host DMA read from the FIFO, if a DSC is loaded into the I/O Buffer Latch the DMA request, DRQOUT, will be deactivated (see Figure 2 above). The Host Status (HSTAT) SFR Data Stream Command bit is set and the INTRQ interrupt output goes active, if enabled. The Host responds to the interrupt as described above.



Symbol	Description	Var. Osc.	@ 16 MHz
TCC	Command Cycle Time	6 * Tcld	375 ns min
TRV	Command Recovery Time	75	75 ns min

Figure 6. UPI-452 Command Cycle Timing

Once INTRQ is deactivated and the DSC has been read by the Host, the DMA request, DRQOUT, is reasserted by the UPI-452. The DMA request then remains active until the transfer is complete or another DSC is loaded into the I/O Buffer Latch.

An Immediate Command written by the internal CPU during a Host DMA FIFO transfer also causes the Host Status flag and INTRQ to go active if enabled. In this case the Immediate Command would not terminate the DMA transfer unless terminated by the Host. The INTRQ line remains active until the Host reads the Host Status (HSTAT) SFR to determine the source of the interrupt.

The net effect of a Data Stream Command (DSC) on DMA data transfer rates is to add an additional factor to the data transfer rate equation. This added factor is shown in Equation 3. An Immediate Command has the same effect on the data transfer rate if the Immediate Command interrupt is recognized by the Host during a DMA transfer. If the DMA transfer is completed before the Immediate Command interrupt is recognized, the effect on the DMA transfer rate depends on whether the block being transmitted is larger than the FIFO channel size. If the block is larger than the programmed FIFO channel size the transfer rate depends on whether the Immediate Command flag or interrupt is recognized between partial block transfers.

The FIFO configuration shown in Equation 3 is arbitrary since there is no way of predicting the size relative to when a DSC would be loaded into the I/O Buffer Latch. The Host DMA rate shown is for a UPI-452

(Memory Mapped or I/O) to 286 System Memory transfer as described earlier. The equations do not account for the latency of initiating the DMA transfer.

Equation 3. Minimum host FIFO DMA Transfer Rate Including Data Stream Command(s)

$$\begin{aligned} \text{Minimum Host/FIFO DMA Transfer Rate w/ DSC} &= \text{FIFO size} * \text{Host DMA 2 cycle time transfer rate} \\ &+ \text{iAPX 286 interrupt response time (Eq. \#1)} \\ &= (32 \text{ bytes} * (200 \text{ ns} + 200 \text{ ns})) + 2.3 \mu\text{s} \\ &= 15.1 \mu\text{s} \\ &= 75.5 \text{ bus cycles (@10 MHz iAPX286, 200 ns bus cycle)} \end{aligned}$$

UPI-452 INTERNAL DMA PROCESSOR

The two identical internal DMA channels allow high speed data transfers from one UPI-452 writable memory space to another. The following UPI-452 memory spaces can be used with internal DMA channels:

- Internal Data Memory (RAM)
- External Data Memory (RAM)
- Special Function Registers (SFR)

The FIFO can be accessed during internal DMA operations by specifying the FIFO IN (FIN) SFR as the DMA Source Address (SAR) or the FIFO OUT (FOUT) SFR as the Destination Address (DAR). Table 6 lists the four types of internal DMA transfers and their respective timings.

4

Table 6. UPI-452 Internal DMA Controller Cycle Timings

Source	Destination	Machine Cycles**	@ 12 MHz	@ 16 MHz
Internal Data Mem. or SFR	Internal Data Mem. or SFR	1	1 μs	750 ns
Internal Data Mem. or SFR	External Data Mem.	1	1 μs	750 ns
External Data Mem.	Internal Data Mem. or SFR	1	1 μs	750 ns
*External Data Memory	External Data Memory	2	2 μs	1.5 μs

NOTES:

*External Data Memory DMA transfer applies to UPI-452 Local Bus only.

**MSC-51 Machine cycle = 12 clock cycles (TCLCL).

FIFO Data Structure and Internal DMA

The effect of Data Stream Commands and Immediate Commands on the internal DMA transfers is essentially the same as the effect on Host FIFO DMA transfers. Recognition also depends upon the programmed DMA Mode, the interrupts enabled, and their priorities. The net internal effect is the same for each possible internal case. The time required to respond to the Immediate or Data Stream Command is a function of the instruction time required. This must be calculated by the user based on the instruction cycle time given in the MSC-51 Instruction Set description in the Intel Microcontroller Handbook.

It is important to note that the internal DMA processor modes and the internal FIFO logic work together to automatically manage internal DMA transfers as data moves into and out of the FIFO. The two most appropriate internal DMA processor modes for the FIFO are FIFO Demand Mode and FIFO Alternate Cycle Mode. In FIFO Demand Mode, once the correct Slave Control and DMA Mode bits are set, the internal Input FIFO channel DMA transfer occurs whenever the Slave Control Input FIFO Request for Service flag is set. The DMA transfer continues until the flag is cleared or when the Input FIFO Read Pointer SFR (IRPR) equals zero. If data continues to be entered by the Host, the internal DMA continues until an internal interrupt of higher priority, if enabled, interrupts the DMA transfer, the internal DMA byte count reaches zero or until the Input FIFO Read Pointer equals zero. A complete description of interrupts and DMA Modes can be found in the UPI-452 Data Sheet.

DMA Modes

The internal DMA processor has four modes of operation. Each DMA channel is software programmable to operate in either Block Mode or Demand Mode. Demand Mode may be further programmed to operate in Burst or Alternate Cycle Mode. Burst Mode causes the internal processor to halt its execution and dedicate its resources exclusively to the DMA transfer. Alternate Cycle Mode causes DMA cycles and instruction cycles to occur alternately. A detailed description of each DMA Mode can be found in the UPI-452 Data Sheet.

INTERFACE LATENCY

The interface latency is the time required to accommodate all of the overhead associated with an individual data transfer. Data transfer rates between the Host system and UPI-452 FIFO, with a block size less than or equal to the programmed FIFO channel size, are calculated using the Host system DMA rate. (see Host DMA description above). In this case, the entire block could be transferred in one operation. The total latency is the time required to accomplish the block DMA transfer, the interrupt response or poll of the Host Status SFR response time, and the time required to initiate the Host DMA processor.

A DMA transfer between the Host and UPI-452 FIFO with a block size greater than the programmed FIFO channel size introduces additional overhead. This additional overhead is from three sources; first, is the time to actually perform the DMA transfer. Second, the overhead of initializing the DMA processor, third, the handshaking between each FIFO block required to transfer the entire data block. This could be time to wait for the FIFO to be emptied and/or the interrupt response time to restart the DMA transfer of the next portion of the block. A fourth component may also be the time required to respond to Underrun and Overrun FIFO Errors.

Table 7 shows six typical FIFO Input/Output channel sizes and the Host DMA transfers times for each. The timings shown reflect a 10 MHz system bus two cycle I/O to Memory DMA transfer rate of 2.5 MBytes/sec as shown in Equation 1. The times given would be the same for iAPX 286 I/O block move instructions REP INS and REP OUTS as described earlier.

Table 7. Host DMA FIFO Data Transfer Times

FIFO Size:	32	43	64	85	96	128	bytes
Full or Empty	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{4}$	Full or Empty	
Time	12.8	17.2	25.6	34.0	38.4	51.2	μ s

Table 8 shows six typical FIFO Input/Output channel sizes and the internal DMA processor data transfers times for each. The timings shown are for a UPI-452 single cycle Burst Mode transfer at 16 MHz or 750 ns per machine cycle in or out of the FIFO channels. The

machine cycle time is that of the MSC-51 CPU; 6 states, 2 XTAL2 clock cycles each or 12 clock cycles per machine cycle. Details on the MSC-51 machine cycle timings and operation may be found in the Intel Microcontroller Handbook.

Table 8. UPI-452 Internal DMA FIFO Data Transfer Times

FIFO Size:	32	43	64	85	96	128	bytes
Full or Empty	1/4	1/3	1/2	2/3	3/4	Full or Empty	
Time	24.0	32.3	48.0	64.6	72.0	96.0	μs

A larger than programmed FIFO channel size data block internal DMA transfer requires internal arbitration. The UPI-452 provides a variety of features which support arbitration between the two internal DMA channels and the FIFO. An example is the internal DMA processor FIFO Demand Mode described above. FIFO Demand Mode DMA transfers occur continuously until the Slave Status Request for Service Flag is deactivated. Demand Mode is especially useful for continuous data transfers requiring immediate attention. FIFO Alternate Cycle Mode provides for interleaving DMA transfers and instruction cycles to achieve a maximum of software flexibility. Both internal DMA channels can be used simultaneously to provide continuous transfer for both Input and Output FIFO channels.

Byte by byte transfers between the FIFO and internal CPU timing is a function of the specific instruction cycle time. Of the 111 MCS-51 instructions, 64 require 12 clock cycles, 45 require 24 clock cycles and 2 require 48 clock cycles. Most instructions involving SFRs are 24 clock cycles except accumulator (for example, MOV direct, A) or logical operations (ANL direct, A). Typical instruction and their timings are shown in Table 9.

Oscillator Period: @ 12 MHz = 83.3 ns
@ 16 MHz = 62.5 ns

Table 9. Typical Instruction Cycle Timings

Instruction	Oscillator Periods	@12 MHz	@16 MHz
MOV direct†, A	12	1 μs	750 ns
MOV direct, direct	24	2 μs	1.5 μs

NOTE:

† Direct = 8-bit internal data locations address. This could be an Internal Data RAM location (0-255) or a SFR [i.e., I/O port, control register, etc.]

Byte by byte FIFO data transfers introduce an additional overhead factor not found in internal DMA operations. This factor is the FIFO block size to be transferred; the number of empty locations in the Output channel, or the number of bytes in the Input FIFO

channel. As described above in the FIFO Data Structure section, the block size would have to be determined by reading the channel read and write pointer and calculating the space available. Another alternative uses the FIFO Overrun and Underrun Error flags to manage the transfers by accepting error flags. In either case the instructions needed have a significant impact on the internal FIFO data transfer rate latency equation.

A typical effective internal FIFO channel transfer rate using internal DMA is shown in Equation 4. Equation 5 shows the latency using byte by byte transfers with an arbitrary factor added for internal CPU block size calculation. These two equations contrast the effective transfer rates when using internal DMA versus individual instructions to transfer 128 bytes. The effective transfer rate is approximately four times as fast using DMA versus using individual instructions (96 μs with DMA versus 492 μs non-DMA).

Equation 4. Effective Internal FIFO Transfer Time Using Internal DMA

$$\begin{aligned} \text{Effective Internal FIFO Transfer Rate with DMA} &= \text{FIFO channel size} * \text{Internal DMA Burst Mode} \\ &\quad \text{Single Cycle DMA Time} \\ &= 128 \text{ Bytes} * 750 \text{ ns} \\ &= 96 \mu\text{s} \end{aligned}$$

Equation 5. Effective FIFO Transfer Time Using Individual Instructions

$$\begin{aligned} \text{Effective Internal FIFO Transfer Rate without DMA} &= \text{FIFO channel size} * \text{Instruction Cycle Time} + \\ &\quad \text{Block size calculation Time} \\ &= 128 \text{ Bytes} * (24 \text{ oscillator periods @ 16 MHz}) + \\ &\quad 20 \text{ instructions (24 oscillator period each @ 16 MHz)} \\ &= 128 * 1.5 \mu\text{s} + 300 \mu\text{s} \\ &= 492 \mu\text{s} \end{aligned}$$

FIFO DMA FREEZE MODE INTERFACE

FIFO DMA Freeze Mode provides a means of locking the Host out of the FIFO Input and Output channels. FIFO DMA Freeze Mode can be invoked for a variety of reasons, for example, to reconfigure the UPI-452 Local Expansion Bus, or change the baud rate on the serial channel. The primary reason the FIFO DMA Freeze Mode is provided is to ensure that the Host does not read from or write to the FIFO while the FIFO interface is being altered. ONLY the internal CPU has the capability of altering the FIFO Special Function Registers, and these SFRs can ONLY be altered during FIFO DMA Freeze Mode. FIFO DMA Freeze Mode inhibits Host access of the FIFO while the internal CPU reconfigures the FIFO.

work while the UPI-452 is in normal operation. Because the external CPU runs asynchronously to the internal CPU, invoking freeze mode without first properly resolving the FIFO Host interface may have serious consequences. Freeze Mode may be invoked only by the internal CPU.

The internal CPU invokes Freeze Mode by setting bit 3 of the Slave Control SFR (SC3). This automatically forces the Slave and Host Status SFR FIFO DMA Freeze Mode to In Progress (SSTAT SSTAT5 = 0, HSTAT SFR HST1 = 1). INTRQ goes active, if enabled by MODE SFR bit 4, whenever FIFO DMA Freeze Mode is invoked to notify the Host. The Host reads the Host Status SFR to determine the source of the interrupt. INTRQ and the Slave and Host Status FIFO DMA Freeze Mode bits are reset by the Host READ of the Host Status SFR.

During FIFO DMA Freeze Mode the Host has access to the Host Status and Control SFRs. All other Host FIFO interface access is inhibited. Table 10 lists the FIFO DMA Freeze Mode status of all slave bus interface Special Function Registers. The internal DMA processor is disabled during FIFO DMA Freeze Mode and the internal CPU has write access to all of the FIFO control SFRs (Table 11).

If FIFO DMA Freeze Mode is invoked without stopping the host, only the last two bytes of data written into or read from the FIFO will be valid. The timing diagram for disabling the FIFO module to the external Host interface is illustrated in Figure 7. Due to this synchronization sequence, the UPI-452 might not go into FIFO DMA Freeze Mode immediately after the Slave Control SFR FIFO 7 DMA Freeze Mode bit (SC3) is set = 0. A special bit in the Slave Status SFR (SSTAT5) is provided to indicate the status of the FIFO DMA Freeze Mode. The FIFO DMA Freeze Mode

SSTAT5 is cleared.

Either the Host or internal CPU can request FIFO DMA Freeze Mode. The first step is to issue an Immediate Command indicating that FIFO DMA Freeze Mode will be invoked. Upon receiving the Immediate Command, the external CPU should complete servicing all pending interrupts and DMA requests, then send an Immediate Command back to the internal CPU acknowledging the FIFO DMA Freeze Mode request. After issuing the first Immediate Command, the internal CPU should not perform any action on the FIFO until FIFO DMA Freeze Mode is invoked. The handshaking is the same in reverse if the Host CPU initiates FIFO DMA Freeze Mode.

After the slave bus interface is frozen, the internal CPU can perform the operations listed below on the FIFO Special Function Registers. These operations are allowed only during FIFO DMA Freeze Mode. Table 11 summarizes the characteristics of all the FIFO Special Function Registers during Normal and FIFO DMA Freeze Modes.

For FIFO Reconfiguration

1. Changing the Channel Boundary Pointer SFR.
2. Changing the Input and Output Threshold SFR.

To Enhance the testability

3. Writing to the read and write pointers of the Input and Output FIFO's.
4. Writing to and reading the Host Control SFRs.
5. Controlling some bits of Host and Slave Status SFRs.
6. Reading the Immediate Command Out SFR and Writing to the Immediate Command in SFR.

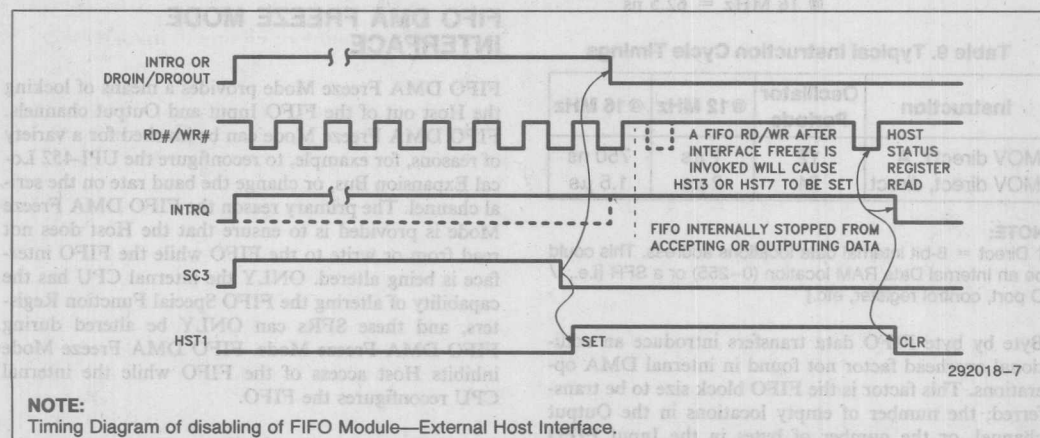


Figure 7. Disabling FIFO to Host Slave Interface Timing Diagram

The sequence of events for invoking FIFO DMA Freeze Mode are listed in Figure 8.

1. Immediate Command to request FIFO DMA Freeze Mode (interrupt)
2. Host/internal CPU interrupt response/service
3. Host/internal CPU clear/service all pending interrupts and FIFO data
4. Internal CPU sets Slave Control (SLCON) FIFO DMA Freeze Mode bit = 0, FIFO DMA Freeze Mode, Host Status SFR FIFO DMA Freeze Mode Status bit = 1, INTRQ active (high)
5. Host READ Host Status SFR
6. Internal CPU reconfigures FIFO SFRs
7. Internal CPU resets Slave Control (SLCON) FIFO DMA Freeze Mode bit = 1, Normal Mode, Host Status FIFO DMA Freeze Mode Status bit = 0.
8. Internal CPU issues Immediate Command to Host indicating that FIFO DMA Freeze Mode is complete
or
Host polls Host Status SFR FIFO DMA Freeze Mode bit to determine end of reconfiguration

Figure 8. Sequence of Events to Invoke FIFO DMA Freeze Mode

EXAMPLE CONFIGURATION

An example of the time required to reconfigure the FIFO 180 degrees, for example from 128 bytes Input to 128 bytes Output, is shown in Figure 9. The example approximates the time based on several assumptions;

1. The FIFO Input channel is full-128 bytes of data
2. Output FIFO channel is empty-1 byte
3. No Data Stream Commands in the FIFO.

4. The Immediate Command interrupt is responded to immediately—highest priority—by Host and internal CPU.

5. Respective interrupt response times

- a. Host (Equation 3 above) = approximately 1.6 μ s
- b. Internal CPU is 86 oscillator periods or approximately 5.38 μ s worst case.

Event	Time
Immediate Command from Host to UPI-452 to request FIFO DMA Freeze Mode (iAPX286 WRITE)	0.30 μ s
Internal CPU interrupt response/service	5.38 μ s
Internal CPU clears FIFO-128 bytes DMA	96.00 μ s
Internal CPU sets Slave Control Freeze Mode bit	0.75 μ s
Immediate Command to Host-Freeze Mode in progress Host Immediate Command interrupt response	2.3 μ s
Internal CPU reconfigures FIFO SFRs	
Channel Boundary Pointer SFR	0.75 μ s
Input Threshold SFR	0.75 μ s
Output Threshold SFR	0.75 μ s
Internal CPU resets Slave Control (SLCON) Freeze Mode bit = 1, Normal Mode, and automatically resets Host Status FIFO DMA Freeze Mode bit	2.3 μ s
Internal CPU writes Immediate Command Out	0.75 μ s
Host Immediate Command interrupt service	2.3 μ s
Total Minimum Time to Reconfigure FIFO	112.33 μ s

Figure 9. Sequence of Events to Invoke FIFO DMA Freeze Mode and Timings

Table 10. Slave Bus Interface Status During FIFO DMA Freezer Mode

Interface Pins;							Operation In	Status In
DACK	CS	A2	A1	A0	READ	WRITE	Normal Mode	Freeze Mode
1	0	0	1	0	0	1	Read Host Status SFR	Operational
1	0	0	1	1	0	1	Read Host Control SFR	Operational
1	0	0	1	1	1	0	Write Host Control SFR	Disabled
1	0	0	0	0	0	1	Data or DMA data from Output Channel	Disabled
1	0	0	0	0	1	0	Data or DMA data to Input Channel	Disabled
1	0	0	0	1	0	1	Data Stream Command from Output Channel	Disabled
1	0	0	0	1	1	0	Data Stream Command to Input Channel	Disabled
1	0	1	0	0	0	1	Read Immediate Command Out from Output Channel	Disabled
1	0	1	0	0	1	0	Write Immediate Command In to Input Channel	Disabled
0	X	X	X	X	0	1	DMA Data from Output Channel	Disabled
0	X	X	X	X	1	0	DMA Data to Input Channel	Disabled

NOTE:
X = don't care

Table 11. FIFO SFR's Characteristics During FIFO DMA Freeze Mode

Label	Name	Normal Operation (SST5 = 1)	Freeze Mode Operation (SST5 = 0)
HCON	Host Control	Not Accessible	Read & Write
HSTAT	Host Status	Read Only	Read & Write
SLCON	Slave Control	Read & Write	Read & Write
SSTAT	Slave Status	Read Only	Read & Write
IEP	Interrupt Enable & Priority	Read & Write	Read & Write
MODE	Mode Register	Read & Write	Read & Write
IWPR	Input FIFO Write Pointer	Read Only	Read & Write
IRPR	Input FIFO Read Pointer	Read Only	Read & Write
OWPR	Output FIFO Write Pointer	Read Only	Read & Write
ORPR	Output FIFO Read Pointer	Read Only	Read & Write
CBP	Channel Boundary Pointer	Read Only	Read & Write
IMIN	Immediate Command In	Read Only	Read & Write
IMONT	Immediate Command Out	Read & Write	Read & Write
FIN	FIFO IN	Read Only	Read Only
CIN	COMMAND IN	Read Only	Read Only
FOUT	FIFO OUT	Read & Write	Read & Write
COUT	COMMAND OUT	Read & Write	Read & Write
ITHR	Input FIFO Threshold	Read Only	Read & Write
OTHR	Other FIFO Threshold	Read Only	Read & Write



APPLICATION NOTE

AP-283

September 1986

4

Flexibility in Frame Size with the 8044

PARVIZ KHODADADI
APPLICATIONS ENGINEER

Order Number: 292019-001

SIZE WITH THE 8044

	PAGE
1.0 INTRODUCTION	4-26
1.1 Normal Operation	4-26
1.2 Expanded Operation	4-27
2.0 THE SERIAL INTERFACE UNIT	4-27
2.1 Hardware Description	4-27
2.2 Reception of Frames	4-28
2.3 Transmission of Frames	4-28
3.0 TRANSMIT AND RECEIVE STATES	4-29
3.1 Receive State Sequence	4-29
3.2 Transmit State Sequence	4-29
4.0 TRANSMISSION/RECEPTION OF LONG FRAMES (Expanded Operation)	4-32
4.1 Description	4-32
4.2 SIU Registers	4-32
4.3 Other Possibilities	4-32
4.4 Maximum Data Rate in Expanded Operation	4-33
5.0 MODES OF OPERATION	4-34
5.1 Flexible Mode	4-34
5.2 Auto Mode	4-34
6.0 APPLICATION EXAMPLES	4-34
6.1 Point-To-Point Application Example ..	4-34
6.1.1 Polling Sequence	4-35
6.1.2 Hardware	4-35
6.1.3 Primary Station Software	4-35
6.1.4 Secondary Station Software	4-39

CONTENTS	PAGE
6.2 Multidrop Application Example	4-42
6.2.1 Polling Sequence	4-42
6.2.2 Hardware	4-42
6.2.3 Primary Station Software	4-44
6.2.4 Secondary Station Software	4-45
6.2.5 Receive Interrupt Routine	4-46
6.2.6 Transmit Subroutine	4-48

The 192 bytes of on-chip RAM serves as the interface buffer between the CPU and the SIU, used by both as a receive and transmit buffer. Some of the internal RAM space is used as general purpose registers (e.g. R0-R7). The remaining bytes may be divided into at least two sections: one section for the transmit buffer and the other section for the receive buffer. In some applications, the 192 byte internal RAM size imposes a limitation on the size of the information field of each frame and, consequently, achieves less than optimal information throughput.

Figure 1 illustrates the flow of data when internal RAM is used as the receive and transmit buffer. The on-chip CPU allocates a receive buffer in the internal RAM and enables the SIU. A receiving SDLC frame is processed by the SIU and the information bytes of the frame, if any, are stored in the internal RAM. Then, the SIU informs the CPU of the received bytes (Serial Channel Interrupt). For transmission, the CPU loads the transmitting bytes into the internal RAM and enables the SIU. The SIU transmits the information bytes in SDLC format.

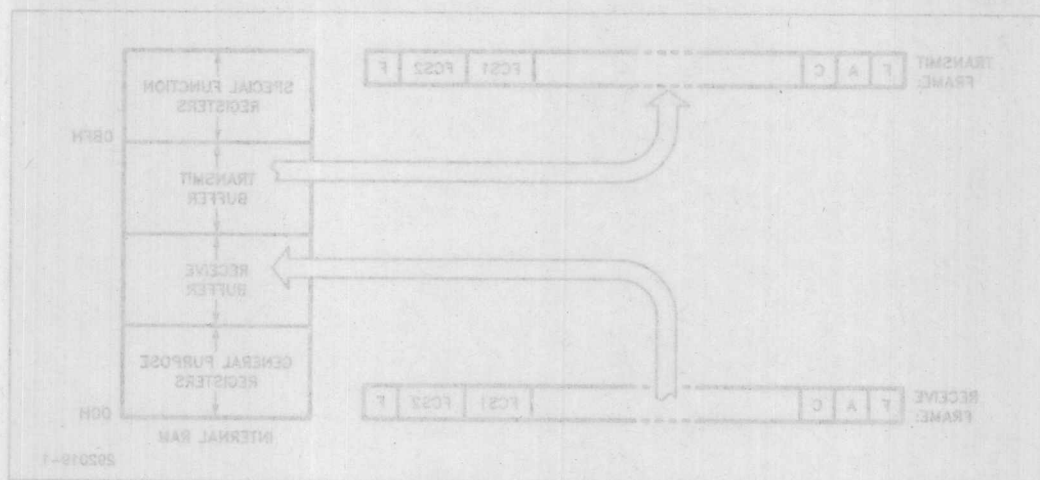


Figure 1. Transmission/Reception Data Flow Using Internal RAM

CONTENTS	PAGE
7.0 CONCLUSIONS	4-48
APPENDIX A - LISTING OF SOFTWARE MODULES FOR APPLICATION EXAMPLE 1	4-49
APPENDIX B - LISTING OF SOFTWARE MODULES FOR APPLICATION EXAMPLE 2	4-51

- (1) Normal operation (limited frame size)
- (2) Expanded operation (unlimited frame size)

In Normal operation the internal 192 byte RAM is used as the receive and transmit buffer. In this operation, the chip supports data rates up to 2.4 Mbps externally clocked and 375 Kbps self-clocked. For frame sizes greater than 192 bytes, Expanded operation is required. In Expanded operation the external RAM, in conjunction with the internal RAM, is used as the transmit and receive buffer. In this operation, the chip supports data rates up to 300 Kbps externally clocked and 375 Kbps self-clocked. In both cases, the SIU handles many of the data link functions in hardware and the chip can be configured in either Auto or Flexible mode.

The discussion that follows describes the operation of the chip and the behavior of the serial interface unit. Both Normal and Expanded operations will be further explained with extra emphasis on Expanded operation and its supporting software. Two examples of SDLC communication systems will also be covered, where the chip is used in Expanded operation. The discussion as-

1.0 INTRODUCTION

The 8044 is a serial communication microcontroller known as the RUPI (Remote Universal Peripheral Interface). It merges the popular 8051 8-bit microcontroller with an intelligent, high performance HDLC/SDLC serial communication controller called the Serial Interface Unit (SIU). The chip provides all features of the microcontroller and supports the Synchronous Data Link Control (SDLC) communications protocol.

There are two methods of operation relating to frame size:

- 1) Normal operation (limited frame size)
- 2) Expanded operation (unlimited frame size)

In Normal operation the internal 192 byte RAM is used as the receive and transmit buffer. In this operation, the chip supports data rates up to 2.4 Mbps externally clocked and 375 Kbps self-clocked. For frame sizes greater than 192 bytes, Expanded operation is required. In Expanded operation the external RAM, in conjunction with the internal RAM, is used as the transmit and receive buffer. In this operation, the chip supports data rates up to 500 Kbps externally clocked and 375 Kbps self-clocked. In both cases, the SIU handles many of the data link functions in hardware, and the chip can be configured in either Auto or Flexible mode.

The discussion that follows describes the operation of the chip and the behavior of the serial interface unit. Both Normal and Expanded operations will be further explained with extra emphasis on Expanded operation and its supporting software. Two examples of SDLC communication systems will also be covered, where the chip is used in Expanded operation. The discussion as-

sumes that the reader is familiar with the 8044 data sheet and the SDLC communications protocol.

1.1 Normal Operation

In Normal operation, the on-chip CPU and the SIU operate in parallel. The SIU handles the serial communication task while the CPU processes the contents of the on-chip transmit and receiver buffer, services interrupt routines, or performs the local real time processing tasks.

The 192 bytes of on-chip RAM serves as the interface buffer between the CPU and the SIU, used by both as a receive and transmit buffer. Some of the internal RAM space is used as general purpose registers (e.g. R0-R7). The remaining bytes may be divided into at least two sections: one section for the transmit buffer and the other section for the receive buffer. In some applications, the 192 byte internal RAM size imposes a limitation on the size of the information field of each frame and, consequently, achieves less than optimal information throughput.

Figure 1 illustrates the flow of data when internal RAM is used as the receive and transmit buffer. The on-chip CPU allocates a receive buffer in the internal RAM and enables the SIU. A receiving SDLC frame is processed by the SIU and the information bytes of the frame, if any, are stored in the internal RAM. Then, the SIU informs the CPU of the received bytes (Serial Channel interrupt). For transmission, the CPU loads the transmitting bytes into the internal RAM and enables the SIU. The SIU transmits the information bytes in SDLC format.

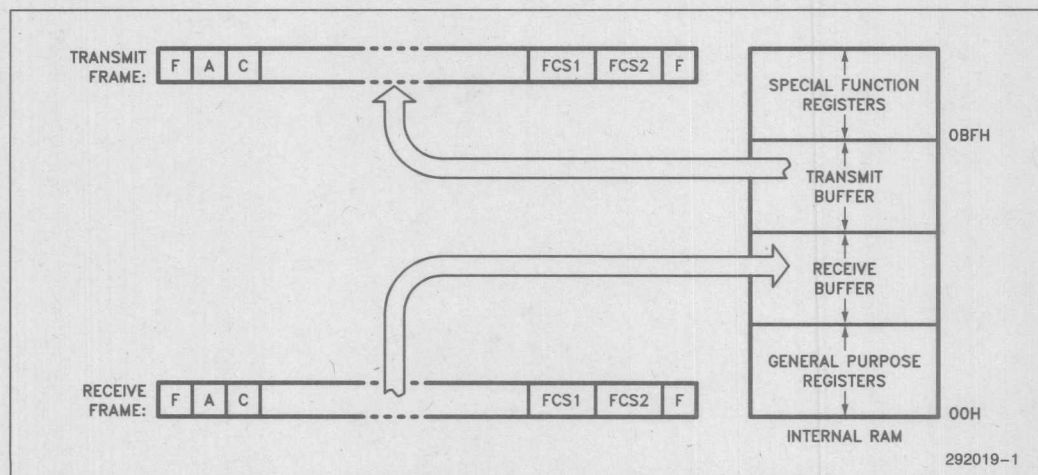


Figure 1. Transmission/Reception Data Flow Using Internal RAM

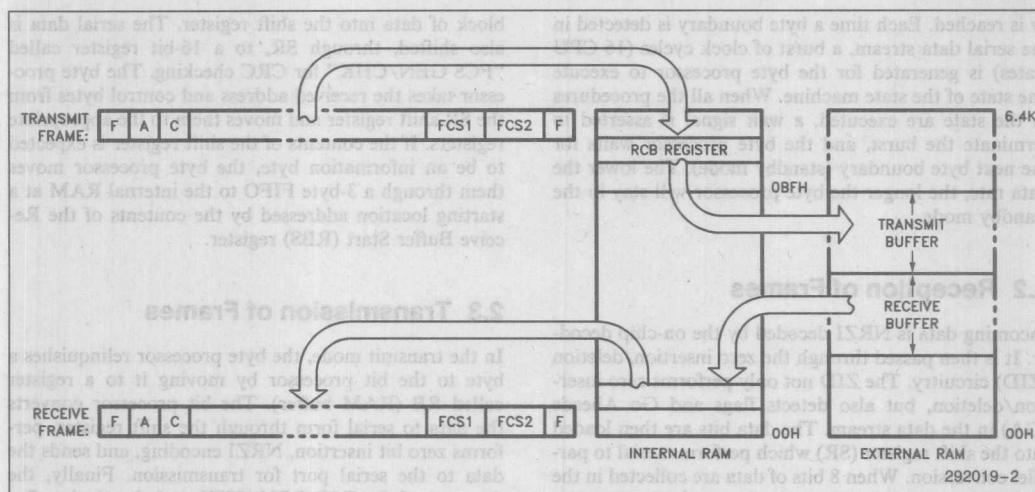


Figure 2. Transmission/Reception Data Flow Using External RAM

1.2 Expanded Operation

In Expanded operation the on-chip CPU monitors the state of the SIU, and moves data from/to external buffer to/from the internal RAM and registers while reception/transmission is taking place. If the CPU must service an interrupt during transmission or reception of a frame or transmit from internal RAM, the chip can shift to Normal operation.

There is a special function register called SIUST, the contents of which dictate the operation of the SIU. Also, at data rates lower than 2.4 Mbps, one section of the SIU, in fixed intervals during transmission and reception, is in the "standby" mode and performs no function. The above two characteristics make it possible to program the CPU to move data to/from external RAM and to force the SIU to repeat or skip some desired hardware tasks while transmission or reception is taking place. With these modifications, external RAM can be utilized as a transmit and receive buffer instead of the internal RAM.

Figure 2 graphically shows the flow of data when external RAM is used. For reception, the receiving bytes are loaded into the Receive Control Byte (RCB) register. Then, the data in RCB is moved to external RAM and the SIU is forced to load the next byte into the RCB register - The chip believes it is receiving a control byte continuously. For transmission, Information bytes (I-bytes) are loaded into a location in the internal RAM and the chip is forced to transmit the contents of this location repeatedly.

Discussion of expanded operation is continued in sections 4 and 5. First, however, sections 2 and 3 describe

features of the 8044 which are necessary to further explain expanded operation.

2.0 THE SERIAL INTERFACE UNIT

2.1 Hardware Description

The Serial Interface Unit (SIU) of the RUPI, shown in Figure 3, is divided functionally into a Bit Processor (BIP) and a Byte Processor (BYP), each sharing some common timing and control logic. The bit processor is the interface between the SIU bus and the serial port pins. It performs all functions necessary to transmit/receive a byte of data to/from the serial data line (shifting, NRZI coding, zero insertion/deletion, etc.). The byte processor manipulates bytes of data to perform message formatting, transmitting, and receiving functions. For example, moving bytes from/to the special function registers to/from the bit processor.

The byte processor is controlled by a Finite-State Machine (FSM). For every receiving/transmitting byte, the byte processor executes one state. It then jumps to the next state or repeats the same state. These states will be explained in section 3. The status of the FSM is kept in an 8-bit register called SIUST (SIU State Counter). This register is used to manipulate the behavior of the byte processor.

As the name implies, the bit processor processes data one bit at a time. The speed of the bit processor is a function of the serial channel data rate. When one byte of data is processed by the bit processor, a byte bounda-

837342

RECEIVED
JAN 22 1988

ss the U

er staff

...and the

to the



Figure 10. The block diagram

3.0 TRANSMIT AND RECEIVE STATES

The simplified receive and transmit state diagrams are shown in Figures 4 and 5, respectively. The numbers on the left of each state represent the contents of the SIUST register when the byte processor is in the standby mode, and the instructions on the right of each state represent the "state procedures" of that state. When the byte processor executes these procedures the least three significant bits of the SIUST register are being incremented while the other bits remain unchanged. The byte processor will jump from one state to another without going into the standby mode when a conditional jump procedure executed by the byte processor is true.

3.1 Receive State Sequence

When an opening flag (7EH) is detected by the bit processor, the byte processor is triggered to execute the procedures of the FLAG state. In the FLAG state, the byte processor loads the contents of the RBS register into the Special RAM (SRAR) register. SRAR is the pointer to the internal RAM. The byte processor decrements the contents of the Receive Buffer Length (RBL) register and loads them into the DMA Count (DCNT) register. The FCS GEN/CHK circuit is turned on to monitor the serial data stream for Frame Check Sequence functions as per SDLC specifications.

Assuming there is an address field in the frame, contents of the SIUST register will then be changed to 08H, causing the byte processor to jump to the ADDRESS state and wait (standby) for the next byte boundary. As soon as the bit processor moves the address byte into the SR shift register, a byte boundary is achieved and the byte processor is triggered to execute the procedures in the ADDRESS state.

In the ADDRESS state the received station address is compared to the contents of the STAD register. If there is no match, or the address is not the broadcast address (FFH), reception will be aborted (SIUST = 01H). Otherwise, the byte processor jumps to the CONTROL state (SIUST = 10H) and goes into standby mode.

The byte processor jumps to the CONTROL state if there exists a control field in the receiving frame. In this state the control byte is moved to the RCB register by the byte processor. Note that the only action taken in this state is that a received byte, processed by the bit processor, is moved to RCB. There is no other hardware task performed, and DCNT and SRAR are not affected in this state.

The next two states, PUSH-1 and PUSH-2, will be executed if Frame check sequence (NFCS = 0) option is selected. In these two states the first and second bytes

of the information field are pushed into the 3-byte FIFO (FIFO0, FIFO1, FIFO2) and the Receive Field Length register (RFL) is set to zero. The 3-byte FIFO is used as a pipeline to move received bytes into the internal RAM. The FIFO prevents transfer of CRC bytes and the closing flag to the receive buffer (i.e., when the ending flag is received, the contents of FIFO are FLAG, FCS1, and FCS0.) The three byte FIFO is collapsed to one byte in No FCS mode.

In the DMA-LOOP state the byte processor pushes a byte from SR to FIFO0, moves the contents of FIFO2 to the internal RAM addressed by the contents of SRAR, increments the SRAR and RFL registers, and decrements the DCNT register. If more information bytes are expected, the byte processor repeats this state on the next byte boundaries until DMA Buffer End occurs. The DMA Buffer End occurs if SRAR reaches 0BFH (192 decimal), DCNT reaches zero, or the RBP bit of the STS register is set.

The BOV-LOOP state, the last state, is executed if there is a buffer overrun. Buffer overrun occurs when the number of information bytes received is larger than the length of the receive buffer ($RFL > RBL$). This state is executed until the closing flag is received.

At the end of reception, if the FCS option is used, the closing flag and the FCS bytes will remain in the 3-byte FIFO. The contents of the RCB register are used to update the NSNR (Receive/Send Count) register. The SIU updates the STS register and sets the serial interrupt.

3.2 Transmit State Sequence

Setting the RTS bit puts the SIU in the transmit mode. When the CTS pin goes active, the byte processor goes into START-XMIT state. In this state the opening flag is moved into the RAM Buffer (RB) register. The byte processor jumps to the next state and goes into the standby mode.

If the Pre-Frame Sync (PFS) option is selected, the PFS1 and PFS2 states will be executed to transmit the two Pre-Frame Sync bytes (00H or 55H). In these two states the contents of the Pre-Frame Sync generator are sent to the serial port while the Zero Insertion Circuit (ZID) is turned off. ZID is turned back on automatically on the next byte boundary.

If the PFS option is not chosen, the byte processor jumps to the FLAG state. In this state, the byte processor moves the contents of TBS into the SRAR register, decrements TBL and moves the contents into the DCNT register. The byte processor turns off the ZID and turns on FCS GEN/CHK. The contents of FCS GEN/CHK are not transmitted unless the NFCS bit is

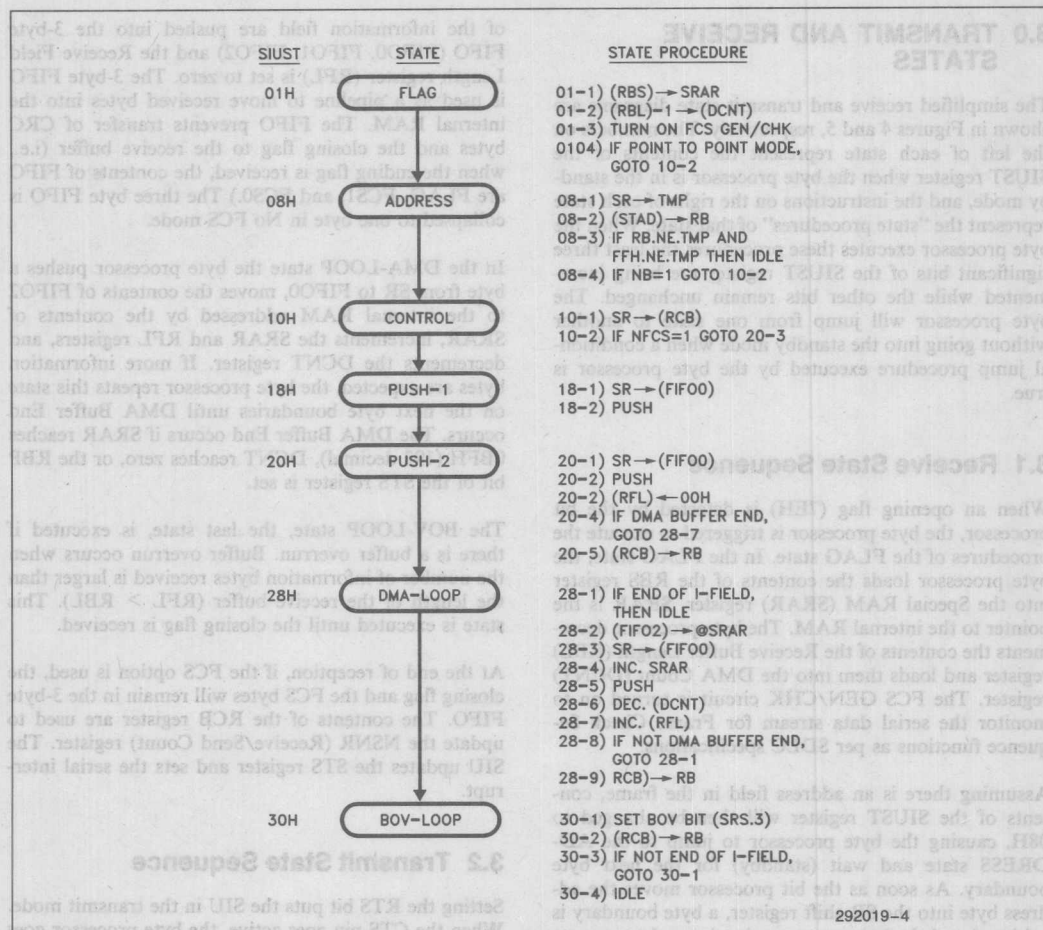


Figure 4. Receive State Diagram

set. If a frame with the address field is chosen, it moves the contents of the STAD register into the RB register for transmission. At the same time, the opening flag is being transmitted by the bit processor.

In the ADDRESS (SIUST = A0H) and CONTROL (SIUST = A8H) states, TCB and the first information byte are loaded into the RB register for transmission, respectively. Note that in the CONTROL state, none of the registers (e.g. DCNT, SRAR) are incremented, and ZID and FCS GEN/CHK are not turned on or off.

The procedures in the DMA-LOOP state are similar to the procedures of the DMA-LOOP in the receive state diagram. The SRAR register pointer to the internal RAM is incremented, and the DCNT register is decremented. The contents of DCNT reach zero when all the information bytes from the transmit buffer are transmitted. A byte from RAM is moved to the RB register for transmission. This state is executed on the following

byte boundaries until all the information bytes are transmitted.

The FCS1 and the FCS2 states are executed to transmit the Frame Check Sequence bytes generated by the FCS generator, and the END-FLAG state is executed to transmit the closing flag.

The XMIT-ACTION and the ABORT-ACTION states are executed by the byte processor to synchronize the SIU with the CPU clock. The XMIT-ACTION or the ABORT-ACTION state is repeated until the byte processor status is updated. At the end, the STS and the TMOD registers are updated.

The two ABORT-SEQUENCE states (SIUST = E0H and SIUST = E8H) are executed only if transmission is aborted by the CPU (RTS or TBF bit of the STS register is cleared) or by the serial data link (CTS signal goes inactive or shut-off occurs in loop mode.)

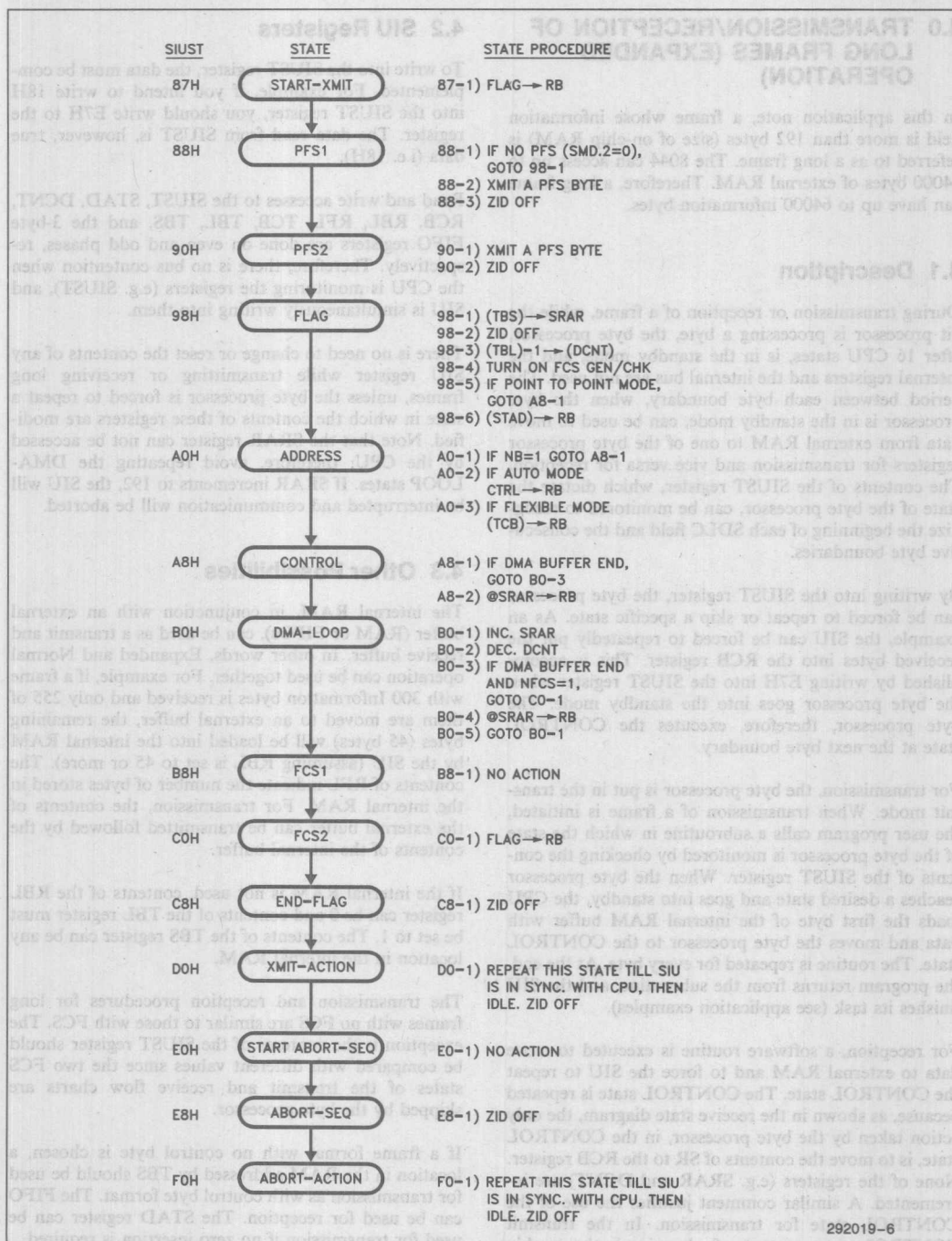


Figure 5. Transmit State Diagram

4.0 TRANSMISSION/RECEPTION OF LONG FRAMES (EXPANDED OPERATION)

In this application note, a frame whose information field is more than 192 bytes (size of on-chip RAM) is referred to as a long frame. The 8044 can access up to 64000 bytes of external RAM. Therefore, a long frame can have up to 64000 information bytes.

4.1 Description

During transmission or reception of a frame, while the bit processor is processing a byte, the byte processor, after 16 CPU states, is in the standby mode, and the internal registers and the internal bus are not used. The period between each byte boundary, when the byte processor is in the standby mode, can be used to move data from external RAM to one of the byte processor registers for transmission and vice versa for reception. The contents of the SIUST register, which dictate the state of the byte processor, can be monitored to recognize the beginning of each SDLC field and the consecutive byte boundaries.

By writing into the SIUST register, the byte processor can be forced to repeat or skip a specific state. As an example, the SIU can be forced to repeatedly put the received bytes into the RCB register. This is accomplished by writing E7H into the SIUST register when the byte processor goes into the standby mode. The byte processor, therefore, executes the CONTROL state at the next byte boundary.

For transmission, the byte processor is put in the transmit mode. When transmission of a frame is initiated, the user program calls a subroutine in which the state of the byte processor is monitored by checking the contents of the SIUST register. When the byte processor reaches a desired state and goes into standby, the CPU loads the first byte of the internal RAM buffer with data and moves the byte processor to the CONTROL state. The routine is repeated for every byte. At the end, the program returns from the subroutine, and the SIU finishes its task (see application examples).

For reception, a software routine is executed to move data to external RAM and to force the SIU to repeat the CONTROL state. The CONTROL state is repeated because, as shown in the receive state diagram, the only action taken by the byte processor, in the CONTROL state, is to move the contents of SR to the RCB register. None of the registers (e.g. SRAR and DCNT) are incremented. A similar comment justifies the use of the CONTROL state for transmission. In the transmit CONTROL state, contents of a location in the on-chip RAM addressed by TBS is moved to RB for transmission.

4.2 SIU Registers

To write into the SIUST register, the data must be complemented. For example, if you intend to write 18H into the SIUST register, you should write E7H to the register. The data read from SIUST is, however, true data (i.e. 18H).

Read and write accesses to the SIUST, STAD, DCNT, RCB, RBL, RFL, TCB, TBL, TBS, and the 3-byte FIFO registers are done on even and odd phases, respectively. Therefore, there is no bus contention when the CPU is monitoring the registers (e.g. SIUST), and SIU is simultaneously writing into them.

There is no need to change or reset the contents of any SIU register while transmitting or receiving long frames, unless the byte processor is forced to repeat a state in which the contents of these registers are modified. Note that the SRAR register can not be accessed by the CPU; therefore, avoid repeating the DMA-LOOP states. If SRAR increments to 192, the SIU will be interrupted and communication will be aborted.

4.3 Other Possibilities

The internal RAM, in conjunction with an external buffer (RAM or FIFOs), can be used as a transmit and receive buffer. In other words, Expanded and Normal operation can be used together. For example, if a frame with 300 Information bytes is received and only 255 of them are moved to an external buffer, the remaining bytes (45 bytes) will be loaded into the internal RAM by the SIU (assuming RBL is set to 45 or more). The contents of RFL indicate the number of bytes stored in the internal RAM. For transmission, the contents of the external buffer can be transmitted followed by the contents of the internal buffer.

If the internal RAM is not used, contents of the RBL register can be 0 and contents of the TBL register must be set to 1. The contents of the TBS register can be any location in the internal RAM.

The transmission and reception procedures for long frames with no FCS are similar to those with FCS. The exception is the contents of the SIUST register should be compared with different values since the two FCS states of the transmit and receive flow charts are skipped by the byte processor.

If a frame format with no control byte is chosen, a location in the RAM addressed by TBS should be used for transmission as with control byte format. The FIFO can be used for reception. The STAD register can be used for transmission if no zero insertion is required.

If the RUPI is used in Auto mode (see Section 5), it will still respond to RR, RNR, REJ, and Unnumbered Poll (UP) SDLC commands with RR or RNR automatically, without using any transmit routine. For example, if the on-chip CPU is busy performing some real time operations, the SIU can transmit an information frame from the internal buffer or transmit a supervisory frame without the help of CPU (Normal operation).

Maximum data rate using this feature is limited primarily by the number of instructions needed to be executed during the standby mode.

Transmission or reception of a frame can be timed out so that the CPU will not hang up in the transmit or receive procedures if a frame is aborted. Or, if the data rate allows enough time (standby time is long enough), the CPU can monitor the SIUST register for idle mode (SIUST = 01H).

It is also possible to transmit multiple opening or closing flags by forcing the byte processor to repeat the END-FLAG state.

4.4 Maximum Data Rate in Expanded Operation

Assuming there is no zero-insertion/deletion, the bit processor requires eight serial clock periods to process one block of data. The byte processor, running on the CPU clock, processes one byte of data in 16 CPU states (one state of the state diagrams). Each CPU state is two oscillator periods. At an oscillator frequency of 12 MHz, the CPU clock is 6 MHz, and 16 CPU states is 2.7 μ s. At a 3 Mbit rate with no zero-insertion/deletion, there is exactly enough time to execute one state per byte (16 states at 6 MHz = 8 bits at 3M baud). In other words, the standby time is zero.

Figure 6 demonstrates portions of the timing relationship between the byte processor and the bit processor. In each state, the actions taken by the processors, plus the contents of the SIUST register, are shown. When the byte processor is running, the contents of SIUST are unknown. However, when it is in the standby mode, its contents are determinable.

The maximum data rate for transmitting and receiving long frames depends on the number of instructions needed to be executed during standby, and is propor-

4

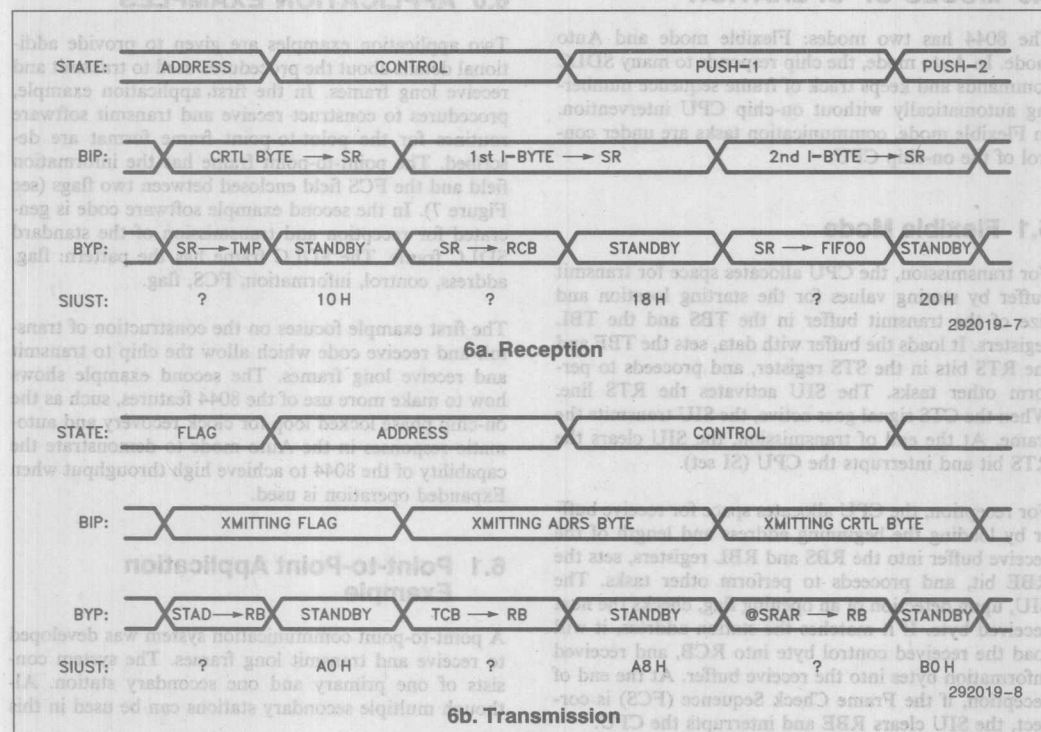


Figure 6. Portions of the BIP/BYP Timing Relationship

tional to the oscillator frequency. The time the byte processor is in the standby mode, waiting for the bit processor to deliver a processed byte, is at least equal to eight serial clock periods minus 16 CPU states. If an inserted zero is in the block of data, the bit processor will process the byte in nine serial clock periods.

The equation for theoretical maximum data rate is given as:

$$(2TCLCL) \times (16 \text{ states}) + (\# \text{ of instruction cycles}) \times (12TCLCL) = (8TDCY) \quad \text{Equation (1)}$$

Where: TCLCL is the oscillator period.
TDCY is the serial clock period.

At an oscillator frequency of 12 MHz and baud rate of 375 Kbps, about 18 instruction cycles can be executed when the byte processor is in the standby mode. At a 9600 baud rate, there is time to execute about 830 instruction cycles—plenty of time to service a long interrupt routine or perform bit-manipulation or arithmetic operations on the data while transmission or reception is taking place.

5.0 MODES OF OPERATION

The 8044 has two modes: Flexible mode and Auto mode. In Auto mode, the chip responds to many SDLC commands and keeps track of frame sequence numbering automatically without on-chip CPU intervention. In Flexible mode, communication tasks are under control of the on-chip CPU.

5.1 Flexible Mode

For transmission, the CPU allocates space for transmit buffer by storing values for the starting location and size of the transmit buffer in the TBS and the TBL registers. It loads the buffer with data, sets the TBF and the RTS bits in the STS register, and proceeds to perform other tasks. The SIU activates the RTS line. When the CTS signal goes active, the SIU transmits the frame. At the end of transmission, the SIU clears the RTS bit and interrupts the CPU (SI set).

For reception, the CPU allocates space for receive buffer by loading the beginning address and length of the receive buffer into the RBS and RBL registers, sets the RBE bit, and proceeds to perform other tasks. The SIU, upon detection of an opening flag, checks the next received byte. If it matches the station address, it will load the received control byte into RCB, and received information bytes into the receive buffer. At the end of reception, if the Frame Check Sequence (FCS) is correct, the SIU clears RBE and interrupts the CPU.

5.2 Auto Mode

In the Auto mode, the 8044 can only be a secondary station operating in the SDLC "Normal Response Mode". The 8044 in Auto mode does not transmit messages unless it is polled by the primary.

For transmission of an information frame, the CPU allocates space for the transmit buffer, loads the buffer with data, and sets the TBF bit. The SIU will transmit the frame when it receives a valid poll-frame. A frame whose poll bit of the control byte is set, is a poll-frame. The poll bit causes the RTS bit to be set. If TBF were not set, the SIU would respond with Receive Not Ready (RNR) SDLC command if RBP = 1, or with Receive Ready (RR) SDLC command if RBP = 0. After transmission RTS is cleared, and the CPU is not interrupted.

For reception, the procedure is the same as that of Flexible mode. In addition, the SIU sets the RTS bit if the received frame is a poll-frame (causing an automatic response) and increments the NS and NR counts accordingly.

6.0 APPLICATION EXAMPLES

Two application examples are given to provide additional details about the procedures used to transmit and receive long frames. In the first application example, procedures to construct receive and transmit software routines for the point-to-point frame format are described. The point-to-point frame has the information field and the FCS field enclosed between two flags (see Figure 7). In the second example software code is generated for reception and transmission of the standard SDLC frame. The SDLC frame has the pattern: flag, address, control, information, FCS, flag.

The first example focuses on the construction of transmit and receive code which allow the chip to transmit and receive long frames. The second example shows how to make more use of the 8044 features, such as the on-chip phase locked loop for clock recovery and automatic responses in the Auto mode to demonstrate the capability of the 8044 to achieve high throughput when Expanded operation is used.

6.1 Point-to-Point Application Example

A point-to-point communication system was developed to receive and transmit long frames. The system consists of one primary and one secondary station. Although multiple secondary stations can be used in this

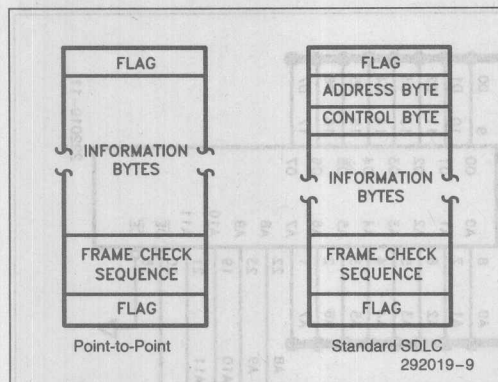


Figure 7. Point-to-Point and Standard SDLC Frame Formats

system, one secondary is chosen to simplify the primary station's software and focus on the long frame software code. Both the primary and the secondary stations are in Flexible mode and the external clock option is used for the serial channel. The maximum data rate is 500 Kbps. The FCS bytes are generated and checked automatically by both stations.

6.1.1 POLLING SEQUENCE

The polling sequence, shown in Figure 8, takes place continuously between the primary and the secondary stations. The primary transmits a frame with one information byte to the secondary. The information byte is used by the secondary as an address byte. The secondary checks the received byte, and if the address matches, the secondary responds with a long frame. In this example, the information field of the frame is chosen to be 255 bytes long. Since there is only one secondary station, the address always matches. Upon successful reception of the long frame, the primary transmits another frame to the secondary station.

6.1.2 HARDWARE

The schematic of the secondary station is given in Figure 9. The circuit of the primary station is identical to the secondary station with the exception of pin 11

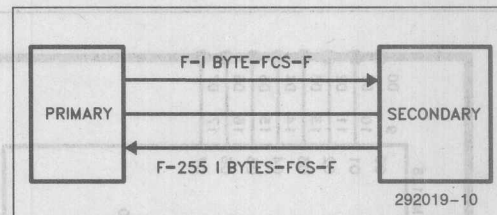


Figure 8. Secondary Responses to Primary Station Commands

(DATA) being connected to pin 14 (T0). In the primary station, the 8044 is interrupted when activity is detected on the communication line by the on-chip timer (in counter mode). This is explained more later. The serial clock to both stations is supplied by a pulse generator. The output of the pulse generator (not shown in the diagram) is connected to pin 15 of the 8044s. Since the two stations are located near each other (less than 4 feet), line drivers are not used.

The central processor of each station is the 8044. The data link program is stored in a 2Kx8 EPROM (2732A), and a 2Kx8 static RAM (AM9128) is used as the external transmit and receive buffer. The RTS pin is connected to the CTS pin. For simplicity, the stations are assumed to be in the SDLC Normal Respond Mode after Hardware reset.

6.1.3 PRIMARY STATION SOFTWARE

The assembly code for the primary station software is listed in Appendix A. The primary software consists of the main routine, the SIU interrupt routine, and the receive interrupt routine. The receive interrupt routine is executed when a long frame is being received.

In the flow charts that follow, all actions taken by the SIU appear in squares, and actions taken by the on-chip CPU appear in spheres.

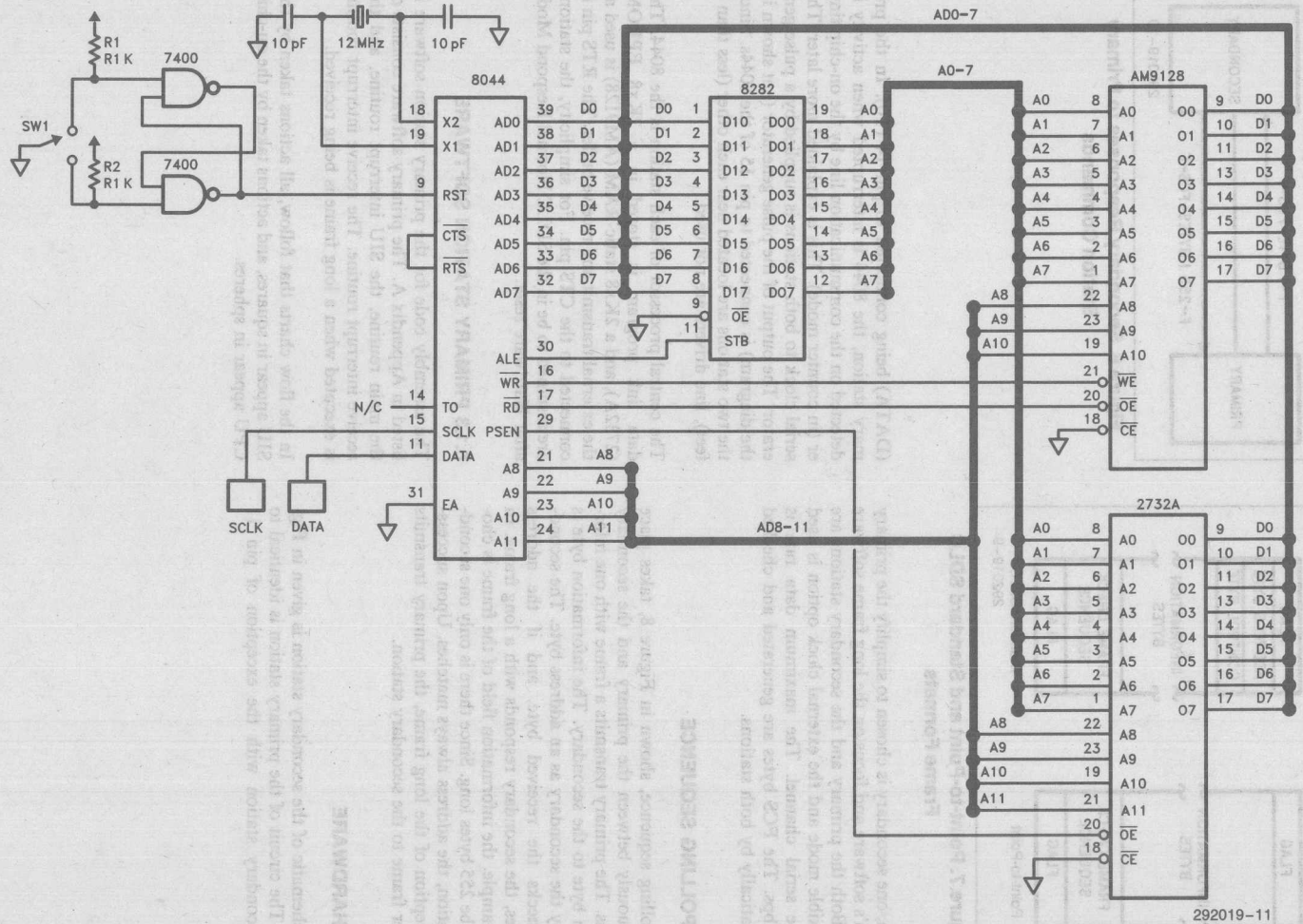


Figure 9. Secondary Station Hardware

Main Routine

First, the chip is initialized (see Figure 10). It is put in Flexible mode, externally clocked, and "Flag-Information Field-FCS-Flag" frame format. Pre-Frame Sync option (PFS = 1) and automatic Frame Check Sequence generation/detection (NFCS = 0) are selected. The on-chip transmit buffer starts at location 20H and the transmit buffer length is set to 1. This one byte buffer contains the address of the secondary station. There is no on-chip receive buffer since the long frame being received is moved to the external buffer. The RTS, TBF, and RBE bits are set simultaneously. Setting the RTS and TBF bits causes the SIU to transmit the contents of the transmit buffer.

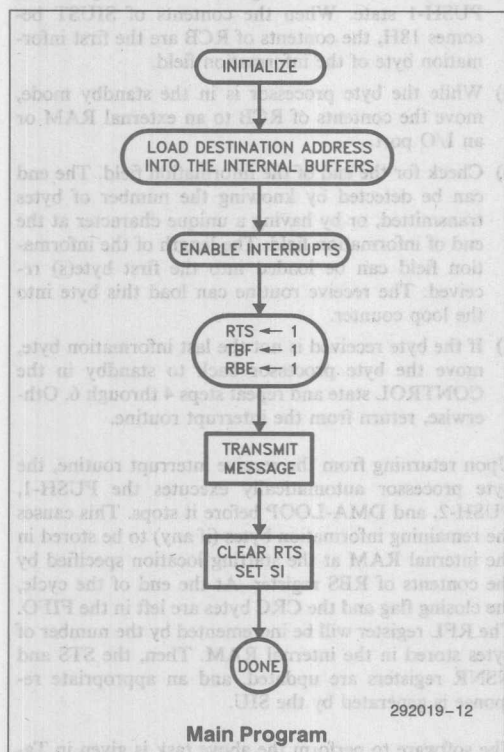


Figure 10. Primary Station Flow Charts

SIU Interrupt Routine

After transmission of the frame, the SIU interrupts the on-chip CPU (SI is set). In the SIU interrupt service routine, counter 0 is initialized and turned on (see Figure 11). The user program returns to perform other

tasks. After reception of the long frame, the SIU interrupt routine is executed again. This time, RTS, TBF, and RBE are set for another round of information exchange between the two stations.

SIU never interrupts while reception or transmission is taking place. The SIU registers are updated and the SI is set (serial interrupt) after the closing flag has been received or transmitted. An SIU interrupt never occurs if the receive interrupt routine or the transmit subroutine is being executed.

Setting the RBE bit of the STS register puts the RUP1 in the receive mode. However, the jump to the receive interrupt routine occurs only when a frame appears on the serial port. Incoming frames can be detected using the Pre-Frame Sync. option and one of the CPU timers in counter mode. The counter external pin (T0) is connected to the data line (pin 11 is tied to pin 14). Setting the PFS (Pre-Frame Sync.) bit will guarantee 16 transitions before the opening flag of a frame.

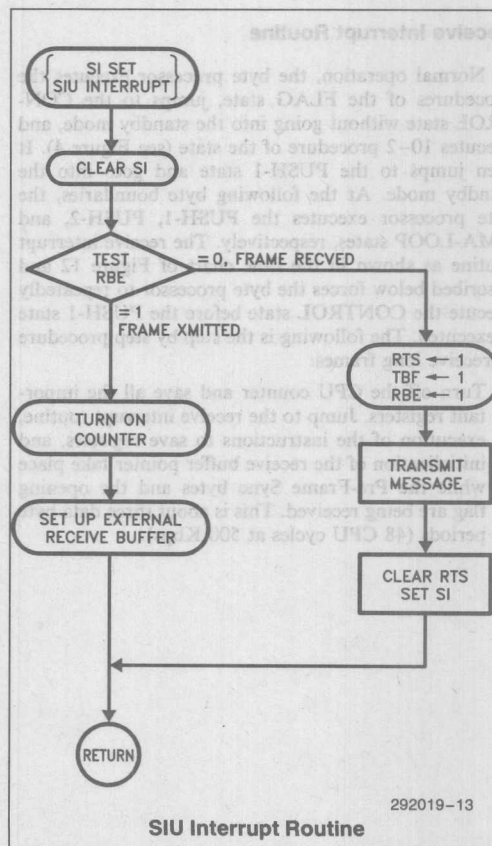


Figure 11. Primary Station Flow Charts

The counter registers are initialized such that the counter interrupt occurs before the opening flag of a frame. When the PFS transitions appear on the data line, the counter overflows and interrupts the CPU. The CPU program jumps to the timer interrupt service routine and executes the receive routine. In the receive routine, the received frame is processed, and the information bytes are moved to the external RAM. Note that the maximum count rate of the 8051 counter is $\frac{1}{24}$ of the oscillator frequency. At 12 MHz, the data rate is limited to 500 Kbps.

Another method to detect a frame on the data line and cause an interrupt is to use an external "Flag-Detect" circuit to interrupt the CPU. The "Flag Detect" circuit can be an 8-bit shift register plus some TTL chips. If this option is used, the RUP1 must operate in externally clocked mode because the clock is needed to shift the incoming data into the shift register. With this option, the maximum data rate is not limited by the maximum count rate of the 8051 counter.

Receive Interrupt Routine

In Normal operation, the byte processor executes the procedures of the FLAG state, jumps to the CONTROL state without going into the standby mode, and executes 10-2 procedure of the state (see Figure 4). It then jumps to the PUSH-1 state and goes into the standby mode. At the following byte boundaries, the byte processor executes the PUSH-1, PUSH-2, and DMA-LOOP states, respectively. The receive interrupt routine as shown in the flow chart of Figure 12 and described below forces the byte processor to repeatedly execute the CONTROL state before the PUSH-1 state is executed. The following is the step by step procedure to receive long frames:

- 1) Turn off the CPU counter and save all the important registers. Jump to the receive interrupt routine, execution of the instructions to save registers, and initialization of the receive buffer pointer take place while the Pre-Frame Sync bytes and the opening flag are being received. This is about three data byte periods (48 CPU cycles at 500 Kbps).

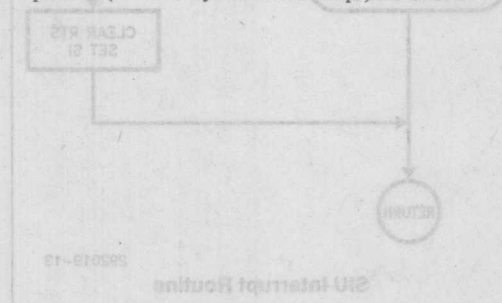


Figure 11: Primary Station Flow Charts

- 2) Monitor the SIUST register for standby in the PUSH-1 state (SIUST = 18H). When the SIUST contents are 18H, the byte processor is waiting for the first information byte. The bit processor has already recognized the flag and is processing the first information byte.
- 3) In the standby mode, move the byte processor into the CONTROL state by writing "EFH" (complement of 10H) into the SIUST register. When the next byte boundary occurs, the bit processor has processed and moved a byte of data into the SR register. The byte processor moves the contents of SR into the RCB register, jumps to the PUSH-1 state (SIUST = 18H), and waits.
- 4) Monitor the SIUST register for standby in the PUSH-1 state. When the contents of SIUST becomes 18H, the contents of RCB are the first information byte of the information field.
- 5) While the byte processor is in the standby mode, move the contents of RCB to an external RAM or an I/O port.
- 6) Check for the end of the information field. The end can be detected by knowing the number of bytes transmitted, or by having a unique character at the end of information field. The length of the information field can be loaded into the first byte(s) received. The receive routine can load this byte into the loop counter.
- 7) If the byte received is not the last information byte, move the byte processor back to standby in the CONTROL state and repeat steps 4 through 6. Otherwise, return from the interrupt routine.

Upon returning from the receive interrupt routine, the byte processor automatically executes the PUSH-1, PUSH-2, and DMA-LOOP before it stops. This causes the remaining information bytes (if any) to be stored in the internal RAM at the starting location specified by the contents of RBS register. At the end of the cycle, the closing flag and the CRC bytes are left in the FIFO. The RFL register will be incremented by the number of bytes stored in the internal RAM. Then, the STS and NSNR registers are updated, and an appropriate response is generated by the SIU.

The software to perform the above task is given in Table 1. In this example, the number of instruction cycles executed during standby is 12 cycles.

Table 1. Codes for Long Frame Reception

Receive Codes		Cycles
REC:	CLR	TRO
WAIT1:	MOV	A, #18H
NEXT1:	CJNE	A, SIUST, WAIT1
	MOV	SIUST, #0EFH
WAIT2:	MOV	A, #18H
	CJNE	A, SIUST, WAIT2
	MOV	A, RCB
	MOVX	@DPTR, A
	INC	DPTR
	DJNZ	R5, NEXTI
	RETI	
END		12 Cycles

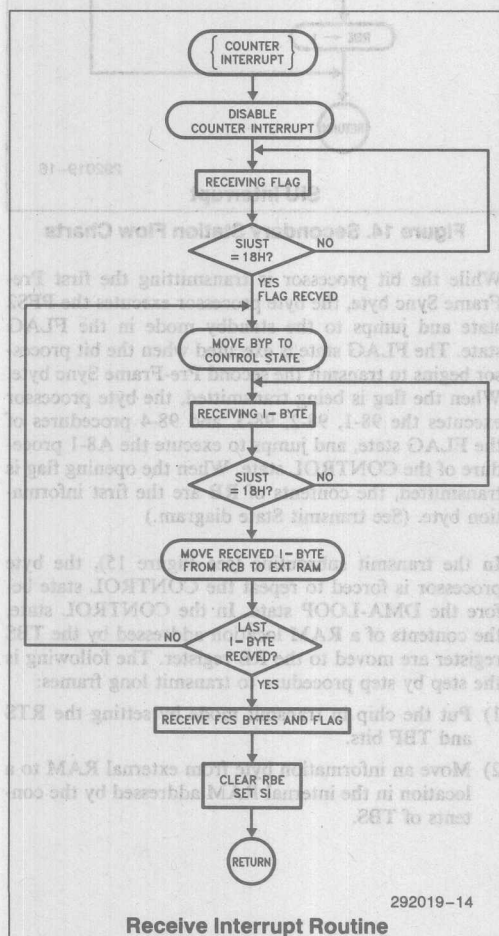


Figure 12. Primary Station Flow Charts

6.1.4 SECONDARY STATION SOFTWARE

The assembly code for the secondary station software is given in Appendix A. The secondary station contains the transmit subroutine which is called for transmission of long frames.

Main Routine

As shown in the secondary station flow chart (Figure 13), the external transmit buffer (external RAM) is loaded with the information data (FFH, FEH, FDH, ...) at starting location 200H. The internal transmit buffer (on chip RAM) starts at location 20H (TBS = 20H), and the transmit buffer length (TBL) is set to 1. The on-chip CPU, in the transmit subroutine, moves the information bytes from the external RAM to this one byte buffer for transmission. The receive buffer starts at location 10H and the receiver buffer length is 1. This buffer is used to buffer the frame transmitted by the primary. The received byte is used as an address byte.

The Secondary is configured like the Primary station. It is put in Flexible mode, externally clocked, Point-to-point frame format. The PFS bit is set to transmit two bytes before the first flag of a frame. The RBE bit is set to put the chip in receive mode. Upon reception of a valid frame, the SIU loads the received information byte into the on-chip receive buffer and interrupts the CPU.

SIU Interrupt Routine

In the serial interrupt routine, the RBE bit is checked (see Figure 14). Since RBE is clear, a frame has been received. The received Information byte is compared with the contents of the Station Address (STAD) register.

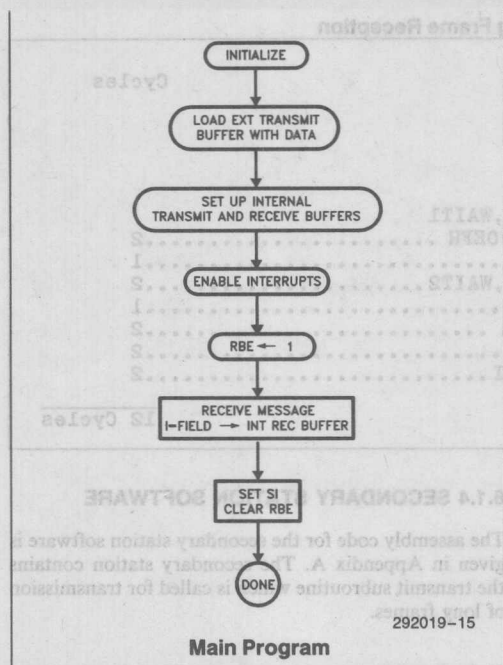


Figure 13. Secondary Station Flow Charts

If they match, the secondary will call the transmit subroutine to transmit the long frame. Upon returning from the transmit subroutine, the RBE bit is set, and program returns from the SIU interrupt. After transmission of the closing flag, SIU interrupt occurs again. In the interrupt routine, the RBE is checked. Since the RBE is set, the program returns from the SIU interrupt routine and waits until another long frame is received.

If the secondary were in Auto mode, the chip must be ready to execute the transmit routine upon reception of a poll-frame; otherwise, the chip automatically transmits the contents of the internal transmit buffer if the TBF bit is set, or transmits a supervisory command (RR or RNR) if TBF is clear.

Transmit Subroutine

In Normal operation the byte processor executes the START-TRANSMIT state and jumps to the PFS1 state. While the bit processor is transmitting some unwanted bits, the byte processor executes the PFS1 state and jumps to the standby mode in the PFS2 state.

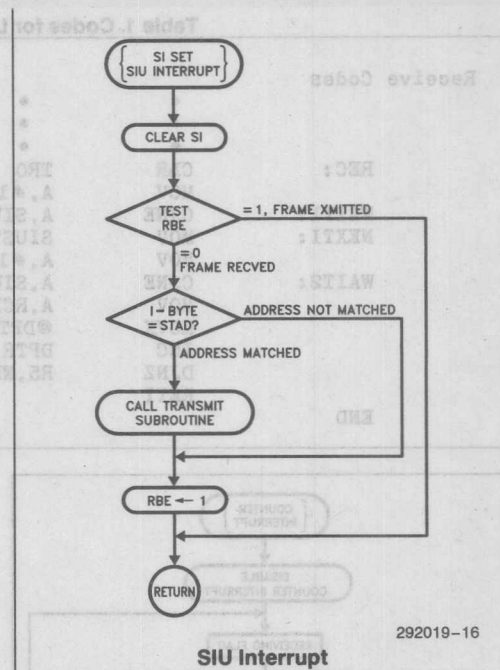


Figure 14. Secondary Station Flow Charts

While the bit processor is transmitting the first Pre-Frame Sync byte, the byte processor executes the PFS2 state and jumps to the standby mode in the FLAG state. The FLAG state is executed when the bit processor begins to transmit the second Pre-Frame Sync byte. When the flag is being transmitted, the byte processor executes the 98-1, 98-2, 98-3, and 98-4 procedures of the FLAG state, and jumps to execute the A8-1 procedure of the CONTROL state. When the opening flag is transmitted, the contents of RB are the first information byte. (See transmit State diagram.)

In the transmit subroutine (see Figure 15), the byte processor is forced to repeat the CONTROL state before the DMA-LOOP state. In the CONTROL state, the contents of a RAM location addressed by the TBS register are moved to the RB register. The following is the step by step procedure to transmit long frames:

- 1) Put the chip in transmit mode by setting the RTS and TBF bits.
- 2) Move an information byte from external RAM to a location in the internal RAM addressed by the contents of TBS.

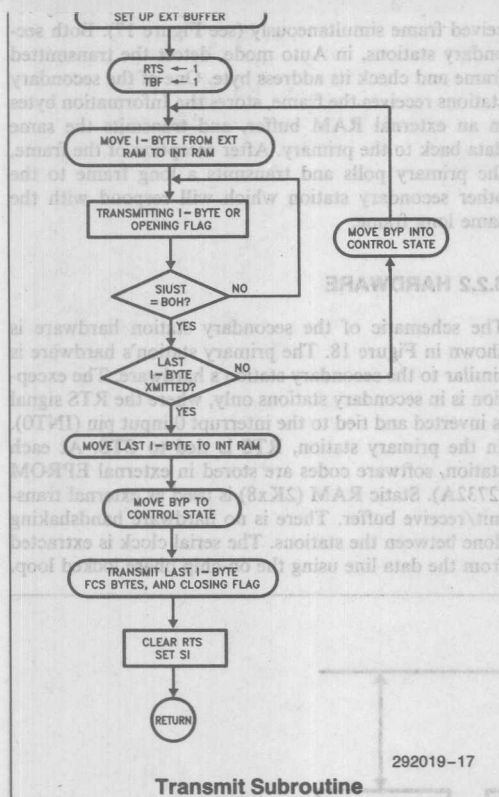


Figure 15. Secondary Station Flow Charts

Table 2. Codes for Long Frame Transmission

Transmit Codes		Cycles
TRAN:	MOV DPTR, #200H	
	MOV R5, #0FFH	
	SETB TBF	
LOOP:	SETB RTS	
	MOVX A, @DPTR	
	MOV @R1, A	
	MOV A, #B0BH	
WAIT1:	CJNE A, SIUST, WAIT1	2
	INC DPTR	2
	MOVX A, @DPTR	2
	MOV @R1, A	1
	DJNZ R5, NEXTI	2
	MOV SIUST, #57H	
NEXTI:	RET	
	MOV SIUST, #57H	2
	MOV A, #B0BH	1
END	JMP WAIT1	1
		13 Cycles

the DMA-LOOP state (SIUST = B0H). When SIUST is B0H, the opening flag has been transmitted, and the first information byte is being transmitted by the bit processor.

- 4) If there are more information bytes, move the byte processor back to the CONTROL state, and repeat steps 2 through 4. Otherwise, continue.
- 5) Move byte processor to the Standby mode in the CONTROL state (SIUST = A8H) and return from the subroutine.

The byte processor automatically executes the remaining states to send the FCS bytes and the closing flag. After the completion of transmission, SIU updates the STS and NSNR registers and interrupts the CPU.

If the contents of the TBL register were more than 1, the SIU transmits (TBL) - 1 additional bytes from the internal RAM at starting address (TBS) + 1 because it executes the DMA-LOOP state (TBL) - 1 additional times. The byte processor should not be programmed to skip the DMA-LOOP state, because the transmission of FCS bytes is enabled in this state.

The maximum baud rate that can be used with these codes is calculated by adding the number of instruction cycles executed, during the standby mode, between each byte boundaries (see Table 2).

Using Equation 1, the maximum data rate, based on the transmit software, is 509 Kbps; However, the maximum count rate of the counter limits the data rate to 500 Kbps.

6.2 Multidrop Application

Performance of long frame in addition to the features of the 8044 are described using a simple multidrop communication system in which three RUPs, one as a master and the other two as secondary stations, transmit and receive long frames alternately (see Figure 16). All stations perform automatic zero bit insertion/deletion, NRZI decoding/encoding, Frame Check Sequence (FCS) generation/detection, and on-chip clock recovery at a data rate of 375 Kbps.

The primary and the secondary station's software code is given in Appendix B. These programs, for simplicity, assume only reception of information and supervisory frames. It is also assumed that the frames are received and transmitted in order. All stations use very similar transmit and receive routines. This code is written for standard SDLC frames (see Figure 7).

6.2.1 POLLING SEQUENCE

The primary station, in Flexible mode, transmits a long frame (for this example, 255 I-bytes), polls one of the

secondary stations, and acknowledges a previously received frame simultaneously (see Figure 17). Both secondary stations, in Auto mode, detect the transmitted frame and check its address byte. One of the secondary stations receives the frame, stores the Information bytes in an external RAM buffer, and transmits the same data back to the primary. After reception of the frame, the primary polls and transmits a long frame to the other secondary station which will respond with the same long frame.

6.2.2 HARDWARE

The schematic of the secondary station hardware is shown in Figure 18. The primary station's hardware is similar to the secondary station's hardware. The exception is in secondary stations only, where the RTS signal is inverted and tied to the interrupt 0 input pin (INT0). In the primary station, RTS is tied to CTS. At each station, software codes are stored in external EPROM (2732A). Static RAM (2Kx8) is used as external transmit/receive buffer. There is no hardware handshaking done between the stations. The serial clock is extracted from the data line using the on-chip phase locked loop.

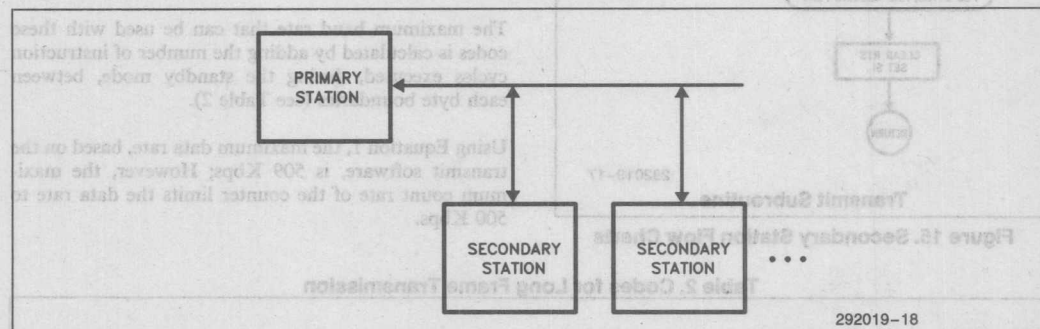


Figure 16. SDLC Multidrop Application Example

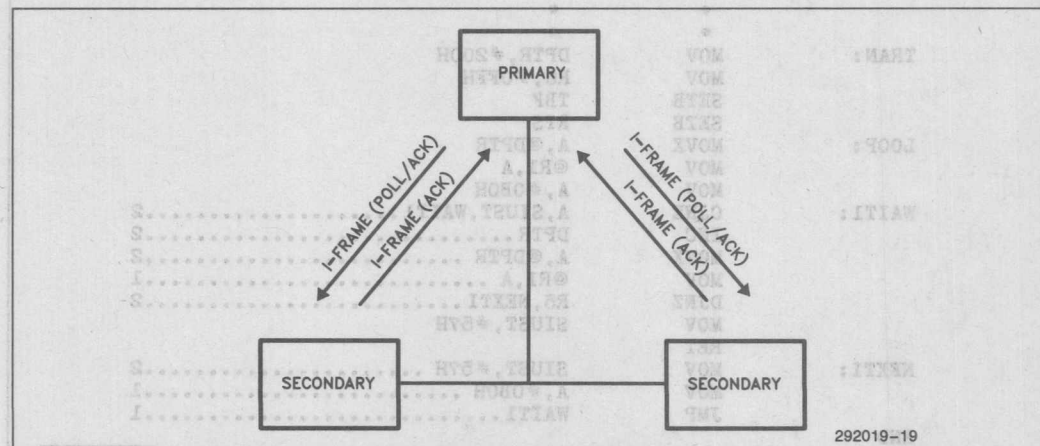
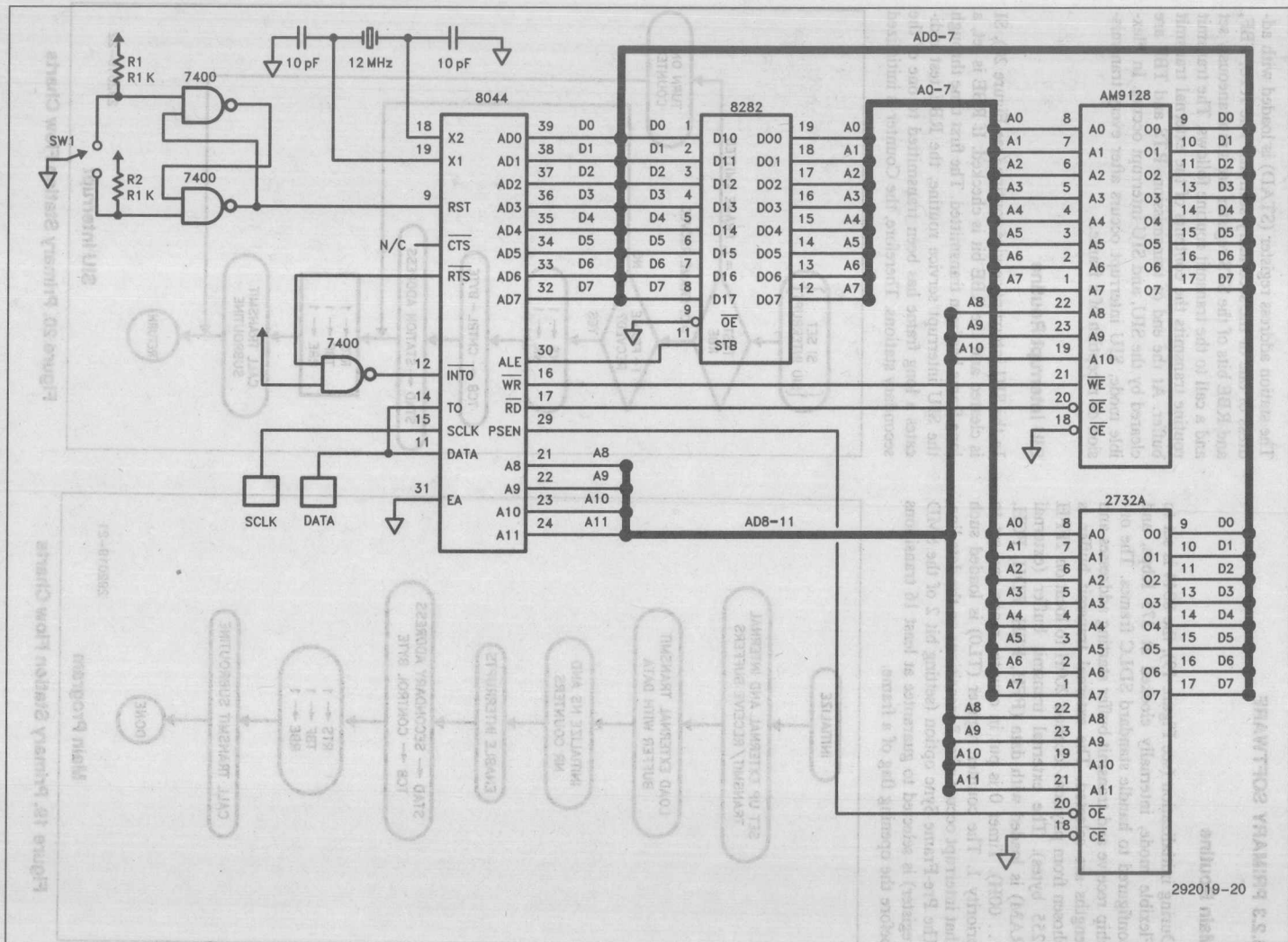


Figure 17. Polling Sequence Between the Primary and Secondary Stations



292019-20

Figure 18. Secondary Station Hardware

6.2.3 PRIMARY SOFTWARE

Main Routine

During initialization (see Figure 19), the 8044 is set to Flexible mode, internally clocked at 375 Kbps, and configured to handle standard SDLC frames. The on-chip receive and transmit buffer starting addresses and lengths are selected. The external transmit buffer is chosen from physical location 200H to location 2FFH (255 bytes). The external transmit buffer (external RAM) is loaded with data (FFH, FEH, FDH, FCH, ... 00H). Timer 0 is put in counter mode and set to priority 1. The counter register (TLO) is loaded such that interrupt occurs after 8 transitions on the data line. The Pre-Frame Sync option (setting bit 2 of the SMD register) is selected to guarantee at least 16 transitions before the opening flag of a frame.

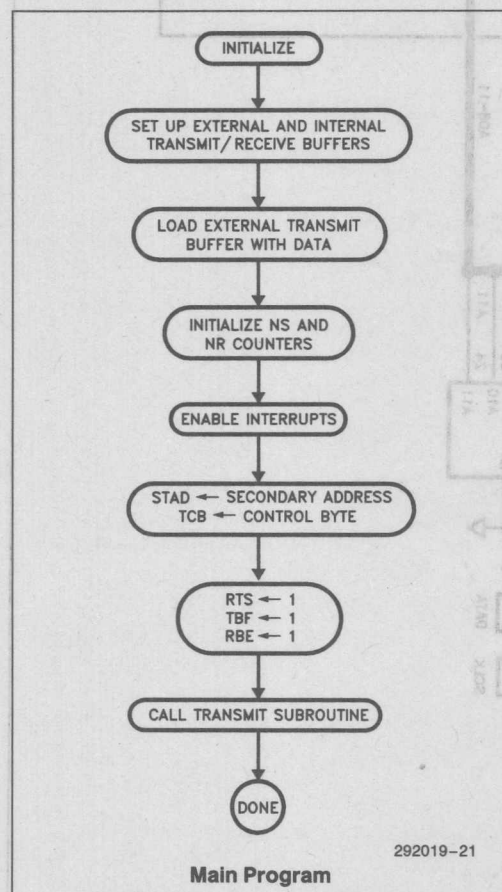


Figure 19. Primary Station Flow Charts

The station address register (STAD) is loaded with address of one of the secondary stations. The RTS, TBF, and RBE bits of the STS register are simultaneously set and a call to the transmit routine follows. The transmit routine transmits the contents of the external transmit buffer. At the end of transmission, RTS and TBF are cleared by the SIU, and SIU interrupt occurs. In Flexible mode, SIU interrupt occurs after every transmission or reception of a frame.

SIU Interrupt Routine

In the SIU interrupt service routine (see Figure 20), SI is cleared and the RBE bit is checked. If RBE is set, a long frame has been transmitted. The first time through the SIU interrupt service routine, the RBE test indicates a long frame has been transmitted to one of the secondary stations. Therefore, the Counter is initialized

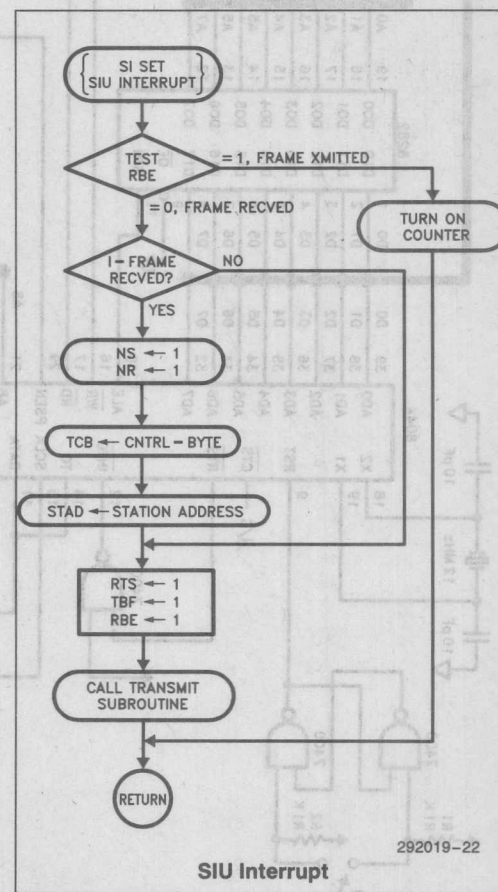


Figure 20. Primary Station Flow Charts

and turned on. The program returns from the interrupt routine before a frame appears on the communication channel.

When a frame appears on the communication line, counter interrupt occurs and the receive routine is executed to move the incoming bytes into the external RAM. After reception of the frame and return from the receive routine, SIU interrupt occurs again.

In the SIU interrupt routine, RBE is checked. Since the RBE bit is clear, a frame has been received. Therefore, the appropriate NS and NR counters are incremented and loaded into the TCB register (two pairs of internal RAM bytes keep track of NS and NR counts for the two secondary stations). Transmission of a frame to the next secondary station is enabled by setting the RTS and the TBF bits. The chip is also put in receive mode (RBE set), and a call to transmit routine is made. After transmission, SIU interrupt occurs again, and the process continues.

6.2.4 SECONDARY SOFTWARE

Main Routine

Both secondary stations have identical software (Appendix B). The only differences are the station addresses. Contents of the STAD register are 55H for one station and 44H for the other.

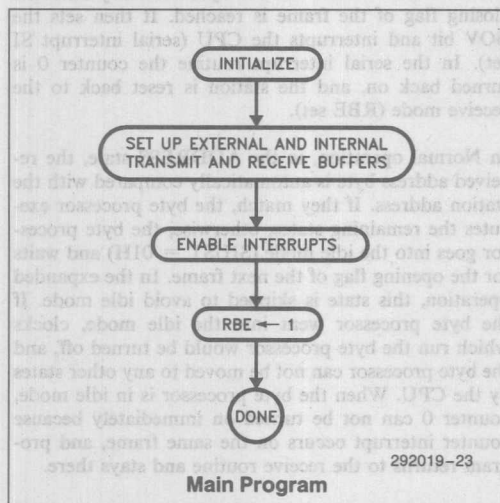


Figure 21. Secondary Station Flow Charts

During initialization, the chip is set to Auto mode, standard SDLC frame, and internally clocked at 375 Kbps (see Figure 21). Internal buffer registers: RBS, RBL, TBS, and TBL are initialized. The RBE bit is set and the counter 0 is turned on.

The secondary is configured to transmit an Information frame every time it is polled. The RTS pin is inverted and tied to INT1 pin. External interrupt 1 is enabled and set to interrupt on low to high transition of the RTS signal. This will cause an interrupt (EX1 set) after a frame is transmitted. In the interrupt routine the CTS pin is cleared to prevent any automatic response from the secondary. If the CTS pin were not disabled, the secondary station would respond with a supervisory frame (RNR) since the TBF is set to zero by the SIU due to the acknowledge. In the SIU interrupt routine, the CTS pin is cleared after the TBF bit is set. If this option is not used, the primary should acknowledge the previously received frame and poll for the next frame in two separate transmissions.

SIU Interrupt Routine

When a frame is received, counter 0 interrupt occurs and the receive routine is executed (see Figure 22). If the incoming frame is addressed to the station, the information bytes are stored in external RAM; Otherwise, the program returns from the receive routine to perform other tasks. At the end of the frame, SIU interrupt occurs. In Auto mode, SIU interrupt occurs whenever an Information frame or a supervisory frame is received. Transmission will not cause an interrupt. In the SIU interrupt service routine, the AM bit of the STS is checked.

If AM bit is set, the interrupt is due to a frame whose address did not match with the address of the station. In this case, NFCS, AM, and the BOV bits are cleared, the RBE bit is set, the counter 0 is initialized and turned on, and program returns from the interrupt routine.

If AM bit is not set, a valid frame has been received and stored in the external RAM. TBF bit is set, CTS pin is activated, counter 0 is disabled and a call to transmit routine is made which transmits the contents of external transmit buffer. This frame also acknowledges the reception of the previously received frame (NS and NR are automatically incremented). Upon return from the transmit routine RBE is set and counter 0 is turned on, thereby putting the chip in the receive mode for another round of data exchange with the primary.

Note that, if the second station is in receive mode, and the counter is enabled and turned on, the CPU will be interrupted each time a frame is on the communication channel. If the frame is not addressed to the secondary station, the chip enters the receive routine, executes only a few lines of code (address comparison) and returns to perform other tasks. This interrupt will not occupy the CPU for more than two data byte periods (43 microseconds at 375 Kbps). At the end of the frame, the BOV bit is set by the SIU, and the SIU interrupt occurs. In the SIU interrupt service routine,

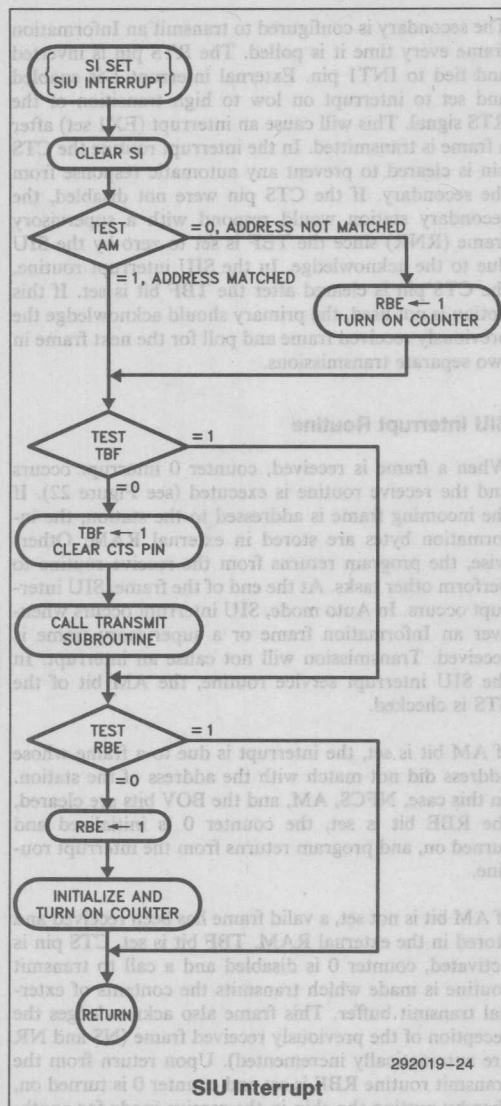


Figure 22. Secondary Station Flow Charts

the RBE bit is set and the counter is turned on which put the chip back in the receive mode.

6.2.5 RECEIVE INTERRUPT ROUTINE

Assembly code for the receive interrupt routine can be found in both primary and secondary software (Appendix B). The receive interrupt routine of the primary station is very similar to that of the primary station in example 1. In the following two sections the receive and transmit routine of the secondary stations are discussed.

In the receive interrupt service routine (see Figure 23), counter 0 is turned off, important registers are saved, receive buffer starting address and receive buffer length of the external RAM are set (do not confuse the external RAM settings with that of the internal RAM buffer.)

After reception of an opening flag, the byte processor jumps to the ADDRESS state and waits until the bit processor processes and moves the receiving address byte to SR. Then, the byte processor is triggered to execute the state. In the secondary stations, the CPU monitors the SIUST register for the ADDRESS state (SIUST = 08H). When the ADDRESS state is reached, the byte processor is moved to the next state (CONTROL state), and the ADDRESS state is skipped. Therefore, when the address byte is moved to SR, the byte processor executes the CONTROL state rather than the ADDRESS state and then jumps to the PUSH-1 state. The execution of the CONTROL state causes the contents of SR (the received address byte) to be loaded into the RCB register.

The CPU checks the contents of RCB with the contents of the STAD (Station Address) register. If they match, the receive routine continues to store the received information bytes in the external RAM buffer; Otherwise, the byte processor is moved to the very last state (BOV-LOOP), and the program returns from the routine to perform other tasks. The byte processor executes the BOV-LOOP state in each byte boundary until the closing flag of the frame is reached. It then sets the BOV bit and interrupts the CPU (serial interrupt SI set). In the serial interrupt routine the counter 0 is turned back on, and the station is reset back to the receive mode (RBE set).

In Normal operation, in the ADDRESS state, the received address byte is automatically compared with the station address. If they match, the byte processor executes the remaining states; otherwise, the byte processor goes into the idle mode (SIUST = 01H) and waits for the opening flag of the next frame. In the expanded operation, this state is skipped to avoid idle mode. If the byte processor went into the idle mode, clocks which run the byte processor would be turned off, and the byte processor can not be moved to any other states by the CPU. When the byte processor is in idle mode, counter 0 can not be turned on immediately because counter interrupt occurs on the same frame, and program returns to the receive routine and stays there.

If the address byte matches the station address, the byte processor is moved to the CONTROL state again. This time, after execution of the CONTROL state the contents of RCB are the received control byte.

CPU investigates the type of received frame by checking the received control byte. If the receiving frame is not an information frame (i.e. Supervisory frame), execution of receive routine will be terminated to free the

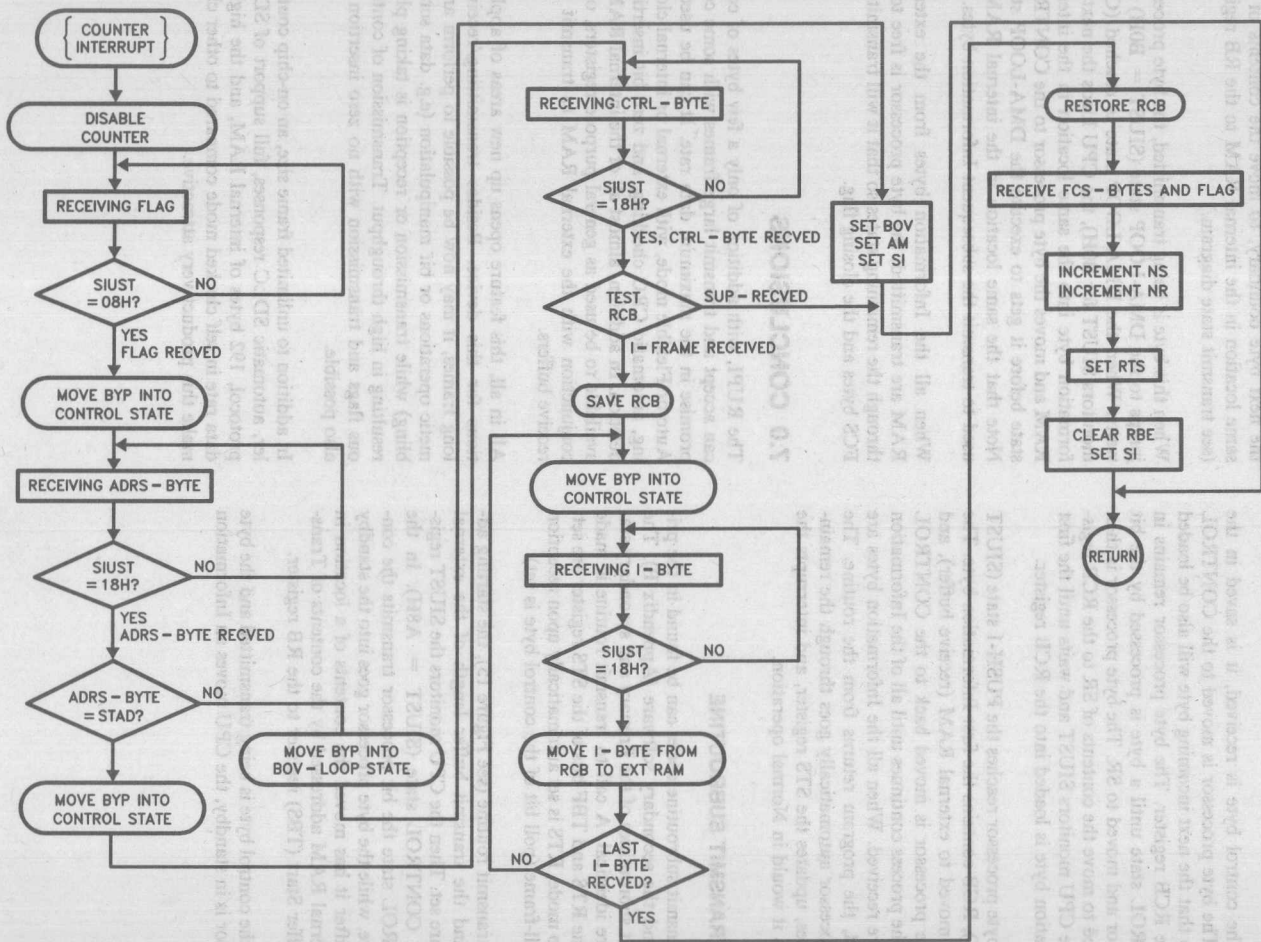


Figure 23. Receive Flow Chart (secondary station)

CPU. In Auto mode, the SIU checks the control byte and responds automatically in response to the supervisory frame.

After the control byte is received, it is saved in the stack. The byte processor is moved to the CONTROL state so that the next incoming byte will also be loaded into the RCB register. The byte processor remains in CONTROL state until a byte is processed by the bit processor and moved to SR. The byte processor is then triggered to move the contents of SR to the RCB register. The CPU monitors SIUST and waits until the first Information byte is loaded into the RCB register.

When byte processor reaches the PUSH-1 state (SIUST = 18H), RCB contains the first Information byte. The byte is moved to external RAM (receive buffer), and the byte processor is moved back to the CONTROL state. The process continues until all of the Information bytes are received. When all the Information bytes are received, the program returns from the routine. The byte processor automatically goes through the remaining states, updates the STS register, and interrupts the CPU as it would in Normal operation.

6.2.6 TRANSMIT SUBROUTINE

The transmit subroutine codes can be found in the primary and the secondary software (Appendix B). The transmit subroutines of the Primary and secondary stations are identical. A call to transmit routine is made when the RTS and TBF bits of the STS register are set. In Auto mode, RTS is set automatically upon reception of a poll-frame (poll bit of the control byte is set).

In the transmit routine (see Figure 15), the starting address and the transmit buffer length of the external buffer are set. Then the CPU monitors the SIUST register for CONTROL state (SIUST = A8H). In the CONTROL state the bit processor transmits the control byte, while the byte processor goes into the standby mode after it has moved the contents of a location in the internal RAM addressed by the contents of Transmit Buffer Start (TBS) register to the RB register.

While the control byte is being transmitted and the byte processor is in standby, the CPU moves an Information

byte from external RAM to the internal RAM location addressed by TBS. The byte processor is then moved to CONTROL state. This will cause the byte processor, in the next byte boundary, to move the contents of the same location in the internal RAM to the RB register (see transmit state diagram.)

When this byte is being transmitted, the byte processor jumps to the DMA-LOOP state (SIUST = B0H) and waits. When the DMA-LOOP state is reached (CPU monitors SIUST for B0H), the CPU loads the next Information byte into the same location in the internal RAM and moves the byte processor to the CONTROL state before it gets to execute the DMA-LOOP state. Note that the same location in the internal RAM is used to transmit the subsequent Information bytes.

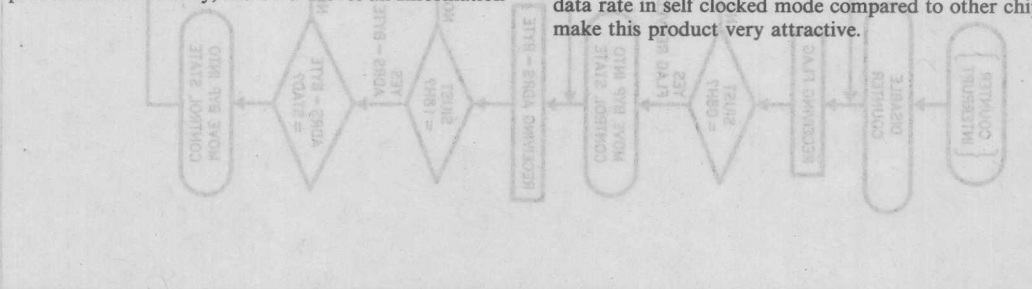
When all the Information bytes from the external RAM are transmitted, the byte processor is free to go through the remaining states so that it will transmit the FCS bytes and the closing flag.

7.0 CONCLUSIONS

The RUPI, with addition of only a few bytes of code, can accept and transmit large frames with some compromise in the maximum data rate. It can be used in Auto or Flexible mode, with external or internal clocking, automatic CRC checking, and zero bit insertion/deletion. In addition, almost all of the internal RAM is available to be used as general purpose registers, or in conjunction with the external RAM as transmit and receive buffers.

All in all, this feature opens up new areas of applications for this device. Besides transmitting/receiving long frames, it may now be possible to perform arithmetic operations or bit manipulation (e.g. data scrambling) while transmission or reception is taking place, resulting in high throughput. Transmission of continuous flags and transmission with no zero insertion are also possible.

In addition to unlimited frame size, an on-chip controller, automatic SDLC responses, full support of SDLC protocol, 192 bytes of internal RAM, and the highest data rate in self clocked mode compared to other chips make this product very attractive.



APPENDIX A

LISTING OF SOFTWARE MODULES FOR APPLICATION EXAMPLE 1

```
$DEBUG NOMOD51
$INCLUDE (REG44.PDF)
```

```
; ASSEMBLY CODE FOR PRIMARY STATION (POINT TO POINT)
; FLEXIBLE MODE; FCS OPTION
```

```
ORG 00H ; LOCATIONS 00 THRU 26H ARE USED
SJMP INIT ; BY INTERRUPT SERVICE ROUTINES.
ORG 0BH ; VECTOR ADDRESS FOR TIMERO INT.
JMP REC ;
ORG 23H ; VECTOR ADDRESS FOR SIU INT.
SJMP SIINT
```

```
;***** INITIALIZATION *****
```

```
INIT: ORG 26H
MOV SMD,#00000110B ; EXT CLOCK; PFS-NB=1
MOV TBS,#20H ; INT TRANSMIT BUFFER START
MOV TBL,#01H ; INT TRANSMIT BUFFER LENGTH
MOV 20H,#55H ; STATION ADDRESS
MOV TMD,#00000111B ; COUNTER FUNCTION; MODE 3
MOV IE,#10010010B ; EA=1; SI=1; ETO=1
MOV STS,#11100000B ; TRANSMIT A FRAME
DOT: SJMP DOT ; WAIT FOR AN INTERRUPT
```

```
; SIU TRANSMITS THE PFS BYTES, THE OPENNING FLAG, THE CONTENTS
; OF LOCATION 20H, THE CALCULATED FCS-BYTES, AND THE CLOSING
; FLAG. AT THE END OF TRANSMISSION, SIU INTERRUPT OCCURS.
```

```
;***** SERIAL CHANNEL INTERRUPT ROUTINE *****
```

```
SIINT: CLR SI
JNB RBE,RCVCD ; TRANSMITTED A FRAME ?
MOV TLO,#0F8H ; YES, INITIALIZE COUNTER REGISTER
MOV DPTR,#200H ; EXT RAM RECEIVE BUFFER START
MOV R5,#0FFH ; EXT RAM RECEIVE BUFFER LENGTH
SETB TR0 ; TURN ON COUNTER 0
RETI ; RETURN
```

```
; WHEN A FRAME APPEARS ON THE SERIAL CHANNEL, COUNTER (RECEIVE)
; INTERRUPT OCCURS. AFTER SERVICING THE INTERRUPT ROUTINE, SIU
; INTERRUPT OCCURS.
```

```
RCVCD: MOV STS,#11100000B ; TRANSMIT A FRAME
RETI ; RETURN
```

```
;***** RECEIVE INTERRUPT ROUTINE *****
```

```
REC: CLR TR0 ; DISABLE THE COUNTER 0 INTERRUPT
MOV A,#19H ; PUSH-1 STATE
WAIT1: CJNE A,S1UST,WAIT1 ;
NEXT1: MOV S1UST,#0EFH ; MOVE BYP TO CONTROL STATE
MOV A,#19H ; PUSH-1 STATE
WAIT2: CJNE A,S1UST,WAIT2 ;
MOV A,RCB ;
MOVX @DPTR,A ; MOVE RECEIVED BYTE INTO ACC.
INC DPTR ; MOVE DATA TO EXT. RAM
DJNZ R5,NEXTI ; INCREMENT POINTER TO EXT RAM
RETI ; LAST BYTE RECEIVED?
; YES, RETURN
```

```
END
```

```
292019-29
```

```
$DEBUG NOMOD51
$INCLUDE (REG44.PDF)

; ASSEMBLY CODE FOR SECONDARY STATION (POINT TO POINT)
; FLEXIBLE MODE; FCS OPTION

ORG 00H
SJMP INIT
ORG 23H
SJMP SIINT
```

***** LOAD TRANSMIT BUFFER WITH DATA *****

```
ORG 26H
INIT: MOV DPTR,#200H ; EXT RAM XMIT BUFFER START
      MOV R3,#0FFH ; EXT RAM XMIT BUFFER LENGTH
LDRAM: MOV A,R3
      MOVX @DPTR,A ; LOAD EXT BUFFER WITH FFH,FEH,...
      INC DPTR ; INCREMENT POINTER
      DJNZ R3,LDRAM
```

*****INITIALIZATION *****

```
MOV SMD,#00000110B ; EXT CLOCK; PFS=NB=1
MOV R1,#10H
MOV TBS,R1 ; INT RAM XMIT BUFFER START
MOV TBL,#01H ; INT RAM XMIT BUFFER LENGTH
MOV RBS,#20H ; INT RAM RECEIVE BUFFER START
MOV RBL,#01H ; INT RAM RECEIVE BUFFER LENGTH
MOV STAD,#55H ; STAD ADDRESS=55H
MOV TCON,#00H ; RESET TCON REGISTER
MOV IE,#10010000B ; ENABLE SI INT. ;EA=1
MOV IP,#0FFH ; ALL INTERRUPTS: PRIORITY 1
MOV STS,#01000000B ; RBE=1, RECEIVE A FRAME.
DOT: SJMP DOT ; WAIT FOR AN INTERRUPT
```

; SIU INTERRUPT OCCURS AT THE END OF A RECEIVED FRAME OR
; A TRANSMITTED FRAME.

***** SERIAL CHANNEL INTERRUPT ROUTINE *****

```
SIINT: CLR SI
      JB RBE,RETRN ; RECEIVED A FRAME?
      MOV A,STAD ; YES
      CJNE A,20H,NMACH ; STATION ADDRESS MATCHED?
      ACALL TRAN ; YES, CALL TRANSMIT SUBROUTINE
```

; TRANSMIT SUBROUTINE IS CALLED TO TRANSMIT A LONG FRAME.
; AFTER TRANSMISSION, SI IS SET. SIU INTERRUPT IS SERVICED
; AFTER THE CURRENT ROUTINE (SIINT) IS COMPLETED.

```
NMACH: SETB RBE ; RBE=1, RECEIVE A FRAME
RETRN: RETI ; RETURN
```

***** TRANSMIT SUBROUTINE *****

```
TRAN: MOV DPTR,#200H ; EXT RAM RECEIVE BUFFER START
      MOV R5,#0FFH ; EXT RAM RECEIVE BUFFER LENGTH
      SETB TBF ; SET TRANSMIT BUFFER FULL
      SETB RTS ; ENABLE XMISSION OF AN I-FRAME
LOOP: MOVX A,@DPTR ; MOVE THE 1ST I-BYTE INTO ACC.
      MOV @R1,A ; THEN, MOVE TO INT. RAM @ (TBS)
      MOV A,#0B0H ; DMA-LOOP STATE
WAIT1: MOV A,SIUST,WAIT1 ; WAIT FOR XMISSION OF AN I-FRAME
      INC DPTR ; INCREMENT POINTER TO EXT. RAM
      DJNZ R5,NEXTI ; ALL BYTES XMITTED?
      MOVX A,@DPTR ; YES, EXCEPT THE LAST BYTE.
      MOV @R1,A ; MOVE DATA INTO INT. RAM @ (TBS)
      MOV SIUST,#57H ; MOVE BYP TO CONTROL STATE
      ; THE SIU TRANSMITS THE FCS-BYTES
      ; AND THE CLOSING FLAG.
      RETN ; RETURN
NEXTI: MOV SIUST,#57H ; MOVE BYP TO CONTROL STATE (ASH)
      JMP LOOP ; TRANSMIT THE NEXT BYTE
END
```

292019-30

292019-31

APPENDIX B

LISTING OF SOFTWARE MODULES FOR APPLICATION EXAMPLE 2

```
$DEBUG NOMOD51
$INCLUDE (REG44.PDF)
```

```
; ASSEMBLY CODE FOR PRIMARY STATION (MULTIPOINT)
; FLEXIBLE MODE; FCS OPTION
```

```
ORG 00H ; LOCATIONS 00 THRU 26H ARE USED
SJMP INIT ; BY INTERRUPT SERVICE ROUTINES.
ORG 0BH ; VECTOR ADDRESS FOR TIMERO INT.
JMP REC ;
ORG 23H ; VECTOR ADDRESS FOR SIU INT.
SJMP SIINT
```

```
;***** LOAD TRANSMIT BUFFER WITH DATA *****
```

```
INIT: ORG 26H ; EXT RAM XMIT BUFFER START
MOV DPTR,#200H ; EXT RAM XMIT BUFFER LENGHT
LDAM: MOV A,R3 ; LOAD BUFFER WITH PFH,FEH,...00
MOVX @DPTR,A ; INCREMENT POINTER
INC DPTR
DJNZ R3,LDAM
```

```
;***** INITIALIZATION *****
```

```
LOOP: MOV RO,#0BFH ; PUT ZEROS INTO INT. RAM
MOV A,#00H ; FROM BFH TO 40H.
MOV @RO,A ; MOVE 0 INTO RAM ADDRESS BY RO
DEC RO ;
CJNE RO,#40H,LOOP ;

MOV 30H,#00H ; NS COUNTER FOR STAD=55
MOV 31H,#00H ; NR COUNTER FOR STAD=55
MOV 32H,#0FFH ; NS COUNTER FOR STAD=44
MOV 33H,#0FFH ; NR COUNTER FOR STAD=44
MOV 34H,#01H ; PONITER TO SECONDARY STATIONS
MOV SMD,#11010100B ; INT. CLKED @ 375K; NRZI=1; PFS=1
MOV RBS,#10H ; INT. RAM RECEIVE BUFFER START=10H
MOV RBL,#00H ; INT. RAM RECEIVE BUFFER LENGTH=0
MOV R1,#20H ; INT. RAM XMIT BUFFER START=20H
MOV TBS,R1
MOV TBL,#01H ; INT. RAM XMIT BUFFER LENGTH=1
MOV NSNR,#00H ; NS=NR=0
MOV TMOD,#00000111B ; COUNTER FUNCTION, MODE 3
MOV TCON,#00H
MOV IE,#10010010B ; EA=1; SI=1; ET0=1
MOV IP,#00000010B ; TIMER 0 INT. PRIORITY 1
MOV TCB,#00010000B ; I-FRAME W/POLL
MOV STAD,#55H ; ADDRESS BYTE=55H
MOV STS,#11100000B ; RBE-TBF-RTS=1
```

```
; TRANSMIT A LONG FRAME WITH POLL BIT SET, WAIT FOR A
; RESPONSE.
```

```
DOT: ACALL TRAN ; CALL TRANSMIT ROUTINE
SJMP DOT ; WAIT FOR AN INTERRUPT
```

292019-32

292019-33

;***** SERIAL INTERRUPT ROUTINE *****

```

SIINT: CLR SI          ; CLEAR SI
       JB RBE,RETURN   ; RECEIVED A FRAME ?
       MOV A,RCB        ; YES, LOAD ACC WITH REC CNTRL BYTE
       JB ACC.0,GETI    ; IS IT AN I-FRAME ?
       MOV A,#01H       ; YES
       CJNE A,34H,SKIP  ;
       MOV A,30H        ; MOVE NS INTO ACC.
       INC A            ; INCREMENT NS
       ANL A,#00000111B ; MASK OUT THE LEAST 3 SIG. BITS
       MOV 30H,A        ; SAVE NS
       MOV A,31H        ; MOVE NR INTO ACC.
       INC A            ; INCREMENT NR
       ANL A,#00000111B ; MASK OUT THE LEAST 3 SIG. BITS
       MOV 31H,A        ; SAVE NR
       RL A             ; SHIFT 4 BITS TO LEFT
       RL A
       RL A
       RL A
       ORL A,30H        ; MOVE NS COUNT TO ACC.
       RL A            ; SHIFT 1 BIT TO LEFT
       ORL A,#00010000B ; SET THE POLL BIT
       MOV TCB,A        ; MOVE CONTROL BYTE INTO TCB REG.

```

```

       MOV STAD,#55H
       MOV 34H,#00H
       JMP GETI

```

```

SKIP:  MOV A,32H        ; MOVE NS INTO ACC.
       INC A            ; INCREMENT NS
       ANL A,#00000111B ; MASK OUT THE LEAST 3 SIG. BITS
       MOV 32H,A        ; SAVE NS
       MOV A,33H        ; MOVE NR INTO ACC.
       INC A            ; INCREMENT NR
       ANL A,#00000111B ; MASK OUT THE LEAST 3 SIG. BITS
       MOV 33H,A        ; SAVE NR
       RL A             ; SHIFT 4 BITS TO LEFT
       RL A
       RL A
       RL A
       ORL A,33H        ; MOVE NS COUNT TO ACC.
       RL A            ; SHIFT 1 BIT TO LEFT
       ORL A,#00010000B ; SET THE POLL BIT
       MOV TCB,A        ; MOVE CONTROL BYTE INTO TCB

```

```

       MOV STAD,#44H
       MOV 34H,#01H
       GETI: MOV STS,#11100000B ; ENABLE TRANSMISSION
       ACALL TRAN        ; CALL TRANSMIT ROUTINE

```

```

       RETI
RETURN: CLR EA          ; DISABLE ALL INTERRUPTS
       MOV TLO,#0FBH    ; INTERRUPT AFTER 8 COUNTS
       SETB TRO         ; TURN ON COUNTER 0
       SETB EA
       RETI

```

;***** RECEIVE INTERRUPT ROUTINE *****

```

REC:   CLR TRO          ; TURN OFF COUNTER 0
       MOV DPTR,#400H   ; EXT. RAM RECEIVE BUFFER START
       MOV R5,#0FFH     ; EXT. RAM RECEIVE BUFFER LENGTH
       MOV A,#18H       ; PUSH-1 STATE
       WAIT1: CJNE A,S1UST,WAIT1 ; WAIT FOR THE CONTROL BYTE
       PUSH RCB          ; SAVE RECEIVE CONTROL BYTE
       NEXT1: MOV SIUST,#0EFH ; PUSH "BYP" INTO CONTROL STATE(10H)
       MOV A,#18H       ; PUSH-1 STATE
       WAIT2: CJNE A,S1UST,WAIT2 ; WAIT FOR AN I-BYTE
       MOV A,RCB        ; MOVE RECEIVED I-BYTE INTO ACC.
       MOVX @DPTR,A     ; MOVE DATA TO EXT. RAM
       INC DPTR          ; INCREMENT PTR TO EXTERNAL RAM
       DJNZ R5,NEXT1    ; IS IT THE LAST I-BYTE?
       POP RCB          ; YES, RESTORE THE CONTENTS OF RCB
       RETI             ; RETURN

```

;***** TRANSMIT SUBROUTINE *****

```

TRAN:  MOV DPTR,#200H   ; EXT. RAM TRANSMIT BUFFER START
       MOV R5,#0FFH     ; EXT. RAM TRANSMIT BUFFER LENGTH
       MOV A,#0A8H      ; CONTROL STATE
       WAIT: CJNE A,S1UST,WAIT ; WAIT FOR CTRL BYTE XMISSION
       MOVX @DPTR,A     ; MOVE DATA FROM EXT. RAM TO ACC.
       MOV @R1,A        ; MOVE DATA INTO INT. RAM @ (TBS)
       INC DPTR          ; INCREMENT POINTER
       DJNZ R5,NXTI     ; IS IT THE LAST I-BYTE ?
       MOV SIUST,#57H   ; NO. XMIT THE LAST I-BYTE
       RET              ; RETURN
       NXTI: MOV SIUST,#57H ; KEEP "BYP" IN CONTROL STATE(A8H)
       MOV A,#0B0H      ; DMA-LOOP STATE
       JMP WAIT         ; TRANSMIT THE NEXT BYTE

```

END

292019-36


```

$DEBUG NOMOD51
$INCLUDE (REG44.PDF)

; ASSEMBLY CODE FOR SECONDARY STATIONS (MULTIPOINT)
; AUTO MODE; FCS OPTION

ORG 00H
SJMP INIT
ORG 0BH
JMP REC
ORG 13H
JMP XINT1
ORG 23H
JMP SIINT

; *****INITIALIZATION *****

INIT:  ORG 26H
      MOV SMD,#11010100B ; INT. CLKED @ 375K;NRZI=1;PFS=1
      MOV STAD,#55H      ; STATION ADDRESS; STAD=44H FOR THE
                        ; OTHER STATION
      MOV RBS,#10H       ; INT. RAM RECEIVE BUFFER START
      MOV RBL,#00H       ; INT. RAM RECEIVE BUFFER LENGTH
      MOV RL,#20H        ; INT. RAM XMIT BUFFER START
      MOV TBS,RL         ; INT. RAM XMIT BUFFER LENGTH
      MOV TBL,#01H       ; NS=NR=0
      MOV NSNR,#00H      ; EXT. INT.: EDGE TRIGGERED
      MOV TCON,#00000100B ; SI=1; ETO=1; EXO=1
      MOV IE,#00010110B  ; TIMER 0: PRIORITY 1
      MOV IP,#00000101B  ; COUNTER FUNCTION: MODE 3
      MOV TMOD,#0000011B ; RECEIVE I-FRAME.
      MOV STS,#01000010B ; SET COUNTER TO OVERFLOW
      MOV TLO,#0F8H      ; AFTER 8 COUNTS
                        ; TURN ON COUNTER
      SETB TRO           ; ENABLE ALL INTERRUPTS
      SETB EA            ; WAIT FOR AN INTERRUPT.
DOT:   SJMP DOT
; CPU IS INTERRUPTED AT THE END OF RECEPTION (SI SET), AND AT*
; THE END OF LONG-FRAME TRANSMISSION (EXO SET).

; *****EXTERNAL INTERRUPT *****

XINT1: SETB P1.7         ; DISABLE CTS PIN
      RETI              ; RETURN.

; ***** SERIAL INTERRUPT ROUTINE *****

SIINT: CLR SI           ; ADDRESS MATCHED?
      JB AM,HOP         ; DISABLE ALL INTERRUPTS
      CLR EA            ; RBE=1; NB=1
      MOV STS,#01000010B
      MOV TLO,#0F8H
      SETB TRO          ; TURN ON COUNTER 0
      SETB EA           ; ENABLE ALL INTERRUPTS
      RETI              ; RETURN.

;
HOP:   JB TBF,GETI      ; A FRAME TRANSMITTED?
      SETB TBF          ; ENABLE TRANSMISSION OF I-FRAME
      CLR P1.7          ; ENABLE CTS PIN
      ACALL TRAN        ; CALL TRANSMIT ROUTINE
GETI:  JB RBE,RETURN    ; A FRAME RECEIVED?
      CLR EA            ; DISABLE ALL INTERRUPTS
      SETB RBE          ; PUT RUPI IN RECEIVE MODE
      MOV TLO,#0F8H
      SETB TRO          ; TURN ON COUNTER 0
      SETB EA           ; ENABLE ALL INTERRUPTS
RETURN: RETI            ; RETURN.

; ***** TRANSMIT SUBROUTINE *****

TRAN:  MOV DPTR,#200H   ; EXT. RAM TRANSMIT BUFFER START
      MOV R5,#0FFH     ; EXT. RAM TRANSMIT BUFFER LENGTH
      MOV A,#0A8H      ; CONTROL STATE
WAIT:  CJNE A,SINST,WAIT ; WAIT FOR CONTROL BYTE TRANSMISSION
      MOVX A,@DPTR      ; MOVE DATA FROM EXT. RAM TO ACC.
      MOV @R1,A         ; MOVE DATA INTO INT. RAM AT @TBS
      INC DPTR          ; INCREMENT POINTER
      DJNZ R5,NXTI      ; IS IT THE LAST I-BYTE ?
      MOV SIUST,#57H    ; XMIT THE LAST I-BYTE
      RET               ; RETURN.
NXTI:  MOV SIUST,#57H   ; KEEP "BYP" IN CONTROL STATE
      MOV A,#0B0H      ; DMA-LOOP STATE
      JMP WAIT          ; TRANSMIT THE NEXT BYTE

```

292019-37

292019-38

292019-39

*****RECEIVE INTERRUPT ROUTINE*****				INSTRUCTION DURING (EXT. RAM) EQUATES	
REC:	CLR	TRO	; TURN OFF COUNTER 0		
	MOV	DPTR,#200H	; EXT. RAM RECEIVE BUFFER START		
	MOV	R5,#0FFH	; EXT. RAM RECEIVE BUFFER LENGTH		
	MOV	A,#06H	; ADDRESS STATE		
HOLD:	CJNE	A,SIUST,HOLD	; WAIT FOR ADDRESS BYTE		
	MOV	SIUST,#0EFH	; MOVE "BYP" INTO CONTROL STATE		
			; SKIP THE ADDRESS STATE		
	MOV	A,#18H	; PUSH-1 STATE		
WAIT1:	CJNE	A,SIUST,WAIT1	; WAIT FOR THE ADDRESS BYTE		
	MOV	A,RCB	; MOVE THE RECEIVED ADDRESS BYTE TO ACC.		
	CJNE	A,STAD,WAIT2	; ADDRESS MATCHED?		
	SJMP	WAIT3	; YES.		
WAIT2:	MOV	RCB,#00010000B	; MOVE INFO. CONTROL BYTE TO RCB		
	MOV	SIUST,#0CFH	; MOVE "BYP" INTO BOV-LOOP STATE		
	RETI		; RETURN		
WAIT3:	MOV	SIUST,#0EFH	; MOVE "BYP" INTO CONTROL STATE		
	MOV	A,#18H	; PUSH-1 STATE		
WAIT4:	CJNE	A,SIUST,WAIT4	; WAIT FOR THE CONTROL BYTE		
	MOV	A,RCB	; MOVE RECEIVE CONTROL BYTE INTO ACC.		
	JB	ACC.0,RTRN	; IF NOT AN I-FRAME RETURN		
	PUSH	RCB	; SAVE RECEIVE CONTROL BYTE		
NEXT1:	MOV	SIUST,#0EFH	; PUSH "BYP" INTO CONTROL STATE(10H).		
	MOV	A,#18H	; PUSH-1 STATE		
WAIT5:	CJNE	A,SIUST,WAIT5	; WAIT FOR AN I-BYTE		
	MOV	A,RCB	; MOVE RECEIVED I-BYTE INTO ACC.		
	MOVB	@DPTR,A	; MOVE DATA TO EXT. RAM		
	INC	DPTR	; INCREMENT PTR TO EXTERNAL RAM		
	DJNZ	R5,NEXT1	; IS IT THE LAST I-BYTE?		
	POP	RCB	; YES. RESTORE THE CONTENTS OF RCB		
RTRN:	RETI		; RETURN		
END					

292019-40

5

80186\188 Application Notes

5

February 1986

High Speed Numerics with the 80186/80188 and 8087

5

STEVE FARRER
APPLICATIONS ENGINEER

Order Number: 231590-001

######

CONTENTS	PAGE
6.0 BENCHMARKS	5-15
6.1 Introduction	5-15
6.2 Interest Rate Calculations	5-15
6.3 Matrix Multiply Benchmark Routine	5-16

Except where noted, all future references to the 80186 will apply equally to the 80188.

2.0 OVERVIEW OF THE 80186

The 80186 and 80188 are highly integrated microprocessors which effectively combine up to 20 of the most common system components onto a single chip. The 80186 and 80188 processors are designed to provide both highest performance and a more highly integrated solution to the total system.

Higher integration results from integrating system peripherals onto the microprocessor. The peripherals consist of a clock generator, an interrupt controller, a DMA controller, a counter/timer unit, a programmable wait state generator, programmable chip selects, and a bus controller. (See Figure 1.)

CONTENTS	PAGE
6.4 Whetstone Benchmark Routine	5-16
6.5 Benchmark Conclusions	5-18
7.0 CONCLUSION	5-18

More and more controller applications need even higher performance in numeric yet still require the low cost and small form factor of the 80186 and 80188. The 8087 Numeric Data Coprocessor satisfies this need as an optional add-on component.

The 8087 Numeric Data Coprocessor is interfaced to the 80186 and 80188 through the 82188 IBC (Integrated Bus Controller). The IBC provides a highly integrated interface solution which replaces the 8255 used in 8086-8087 systems. The IBC incorporates all the necessary bus control for the 8087 while also providing the necessary logic to support the interface between the 80186/87 and the 8087.

This application note discusses the design considerations associated with using the 8087 Numeric Data Coprocessor with the 80186 and 80188. Sections two

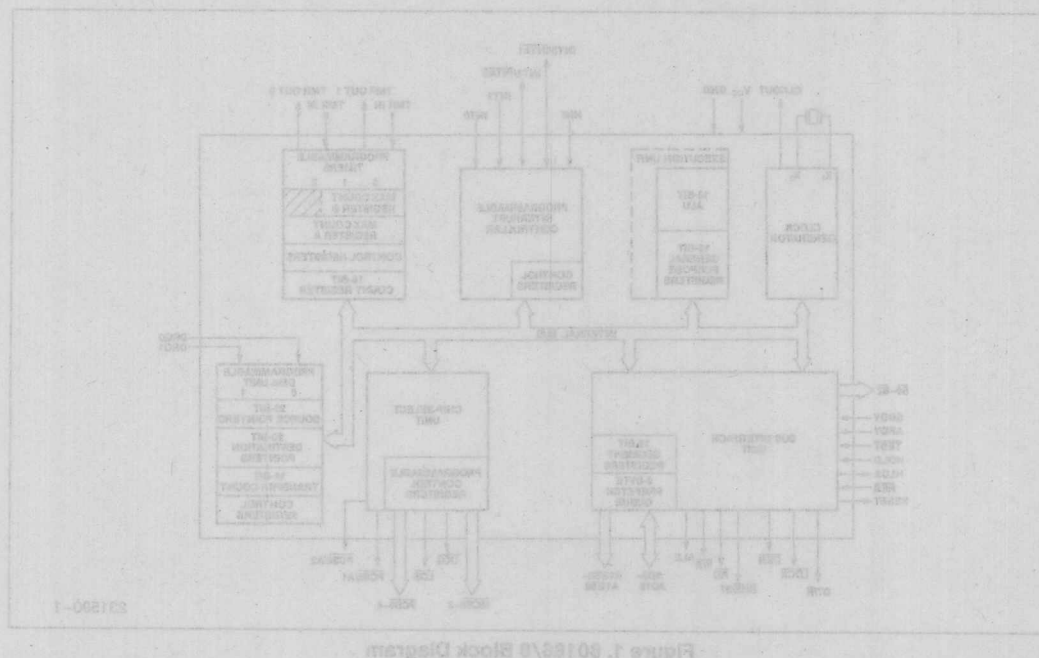


Figure 1. 80186/8 Block Diagram

1.0 INTRODUCTION

From their introduction in 1982, the highly integrated 16-bit 80186 and its 8-bit external bus version, the 80188, have been ideal processor choices for high-performance, low-cost embedded control applications. The integrated peripheral functions and enhanced 8086 CPU of the 80186 and 80188 allow for an easy upgrade of older generation control applications to achieve higher performance while lowering the overall system cost through reduced board space, and a simplified production flow.

More and more controller applications need even higher performance in numerics, yet still require the low-cost and small form factor of the 80186 and 80188. The 8087 Numerics Data Coprocessor satisfies this need as an optional add-on component.

The 8087 Numeric Data Coprocessor is interfaced to the 80186 and 80188 through the 82188 IBC (Integrated Bus Controller). The IBC provides a highly integrated interface solution which replaces the 8288 used in 8086-8087 systems. The IBC incorporates all the necessary bus control for the 8087 while also providing the necessary logic to support the interface between the 80186/8 and the 8087.

This application note discusses the design considerations associated with using the 8087 Numeric Data Coprocessor with the 80186 and 80188. Sections two,

three, and four contain an overview of the integrated circuits involved in the numerics configuration. Section five discusses the interfacing aspects between the 80186/8 and the 8087, including the role of the 82188 Integrated Bus Controller and the operation of the integrated peripherals on the 80186/8 with the 8087. Section six compares the advantages of using an 8087 Numeric Data Coprocessor over software routines written for the host processor as well as the advantage of using an 80186/8 numerics system over an 8086/8088 numerics system.

Except where noted, all future references to the 80186 will apply equally to the 80188.

2.0 OVERVIEW OF THE 80186

The 80186 and 80188 are highly integrated microprocessors which effectively combine up to 20 of the most common system components onto a single chip. The 80186 and 80188 processors are designed to provide both higher performance and a more highly integrated solution to the total system.

Higher integration results from integrating system peripherals onto the microprocessor. The peripherals consist of a clock generator, an interrupt controller, a DMA controller, a counter/timer unit, a programmable wait state generator, programmable chip selects, and a bus controller. (See Figure 1.)

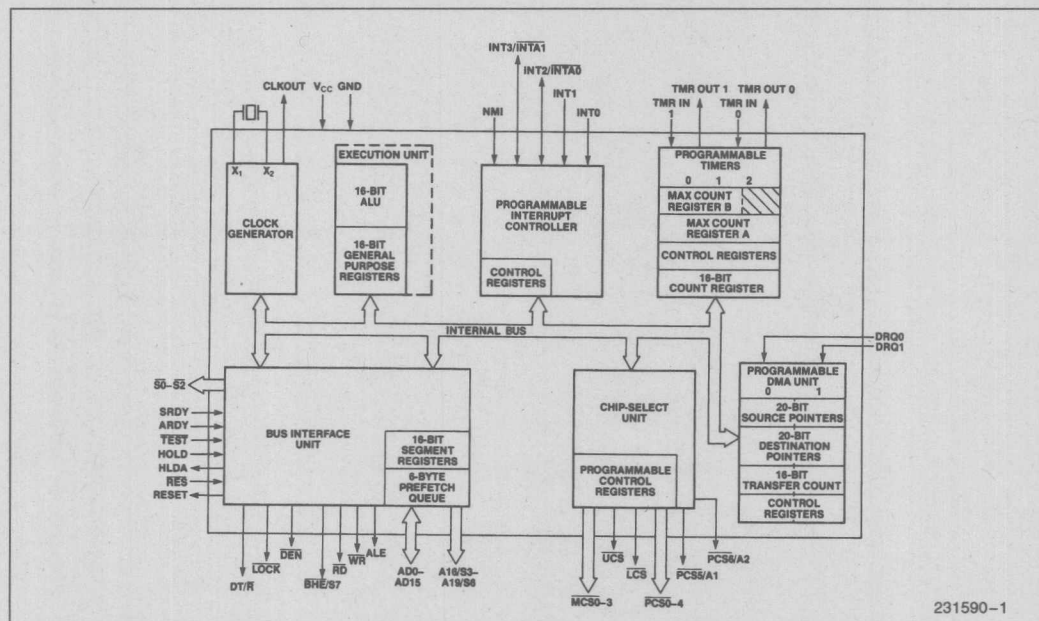


Figure 1. 80186/8 Block Diagram

Higher performance results from enhancements to both general and specific areas of the 8086 CPU, including faster effective address calculation, improvement in the execution speed of many instructions, and the inclusion of new instructions which are designed to produce optimum 80186 code.

The 80186 and 80188 are completely object code compatible with the 8086 and 8088. They have the same basic register set, memory organization, and addressing modes. The differences between the 80186 and 80188 are the same as the differences between the 8086 and 8088: the 80186 has a 16-bit architecture and 16-bit bus interface; the 80188 has a 16-bit internal architecture and an 8-bit data bus interface. The instruction execution times of the two processors differ accordingly: for each non-immediate 16-bit data read/write instruction, 4 additional clock cycles are required by the 80188.

3.0 NUMERICS OVERVIEW

3.1 The Benefits of Numeric Coprocessing

The 8086/8 and 80186/8 are general purpose microprocessors, designed for a very wide range of applications. Typically, these applications need fast, efficient data movement and general purpose control instructions. Arithmetic on data values tends to be simple in these applications. The 8086/8 and 80186/8 fulfill these needs in a low cost, effective manner.

However, some applications require extremely fast and complex math functions which are not provided by a general purpose processor. Such functions as square root, sine, cosine, and logarithms are not directly available in a general purpose processor. Software routines required to implement these functions tend to be slow and not very accurate. Integer data types and their arithmetic operations (i.e., add, subtract, multiply and divide) which are directly available on general purpose processors, still may not meet the needs for accuracy, speed and ease of use.

Providing fast, accurate, complex math can be quite complicated, requiring large areas of silicon on integrated circuits. A general data processor does not provide these features due to the extra cost burden that less complex general applications must take on. For such features, a special numeric data processor is required — one which is easy to use and has a high level of support in hardware and software.

3.2 Introduction to the 8087

The 8087 is a numeric data coprocessor which is capable of performing complex mathematical functions while the host processor (i.e. the main CPU) performs

more general tasks. It supports the necessary data types and operations and allows use of all the current hardware and software support for the 8086/8 and 80186/8 microprocessors. The fact that the 8087 is a coprocessor means it is capable of operating in parallel with the host CPU, which greatly improves the processing power of the system.

The 8087 can increase the performance of floating-point calculations by 50 to 100 times, providing the performance and precision required for small business and graphics applications as well as scientific data processing.

The 8087 numeric coprocessor adds 68 floating-point instructions and eight 80-bit floating-point registers to the basic 8086 programming architecture. All the numeric instructions and data types of the 8087 are used by the programmer in the same manner as the general data types and instructions of the host.

The numeric data formats and arithmetic operations provided by the 8087 support the proposed IEEE Microprocessor Floating Point Standard. All of the proposed IEEE floating point standard algorithms, exception detection, exception handling, infinity arithmetic and rounding controls are implemented. The IEEE standard makes it easier to use floating point and helps to avoid common problems that are inherent to floating point.

3.3 Escape Instructions

The coprocessing capabilities of the 8087 are achieved by monitoring the local bus of the host processor. Certain instructions within the 8086 assembly language known as ESCAPE instructions are defined to be coprocessor instructions and, as such, are treated differently.

The coprocessor monitors program execution of the host processor to detect the occurrence of an ESCAPE instruction. The fetching of instructions is monitored via the data bus and bus cycle status S2-S0, while the execution of instructions is monitored via the queue status lines QS0 and QS1.

All ESCAPE instructions start with the high-order 5-bits of the instruction opcode being 11011. They have two basic forms, the memory reference form and the non-memory reference form. The non-memory form, shown in Figure 2A, initiates some activity in the coprocessor using the nine available bits of the ESCAPE instruction to indicate which function to perform.

Memory reference forms of the ESCAPE instruction, shown in Figure 2B, allow the host to point out a memory operand to the coprocessor using any host memory

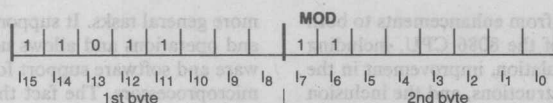


Figure 2A. Non-Memory Reference ESCAPE Instructions

addressing mode. Six bits are available in the memory reference form to identify what to do with the memory operand.

Memory reference forms of ESCAPE instructions are identified by bits 7 and 6 of the byte following the ESCAPE opcode. These two bits are the MOD field of the 8086/8 or 80186/8 effective address calculation byte. Together with the R/M field (bits 2 through 0), they determine the addressing mode and how many subsequent bytes remain in the instruction.

3.4 Host Response to Escape Instructions

The host performs one of two possible actions when encountering an ESCAPE instruction: do nothing (operation is internal to 8087) or calculate an effective address and read a word value beginning at that address (required for all LOADS and STORES). The host ignores the value of the word read and hence the cycle is referred to as a "Dummy Read Cycle." ESCAPE instructions do not change any registers in the host other than advancing the IP. If there is no coprocessor or the coprocessor ignores the ESCAPE instruction, the ESCAPE instruction is effectively a NOP to the host. Other than calculating a memory address and reading a word of memory, the host makes no other assumptions regarding coprocessor activity.

The memory reference ESCAPE instructions have two purposes: to identify a memory operand and, for certain instructions, to transfer a word from memory to the coprocessor.

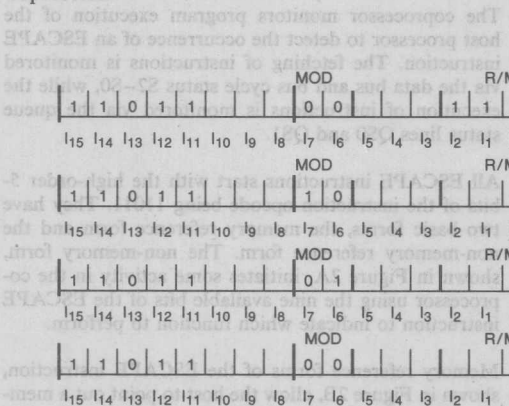


Figure 2B. Memory Reference ESCAPE Instruction Forms

3.5 Coprocessor Response to Escape Instructions

The 8087 performs basically three types of functions when encountering an ESCAPE instruction: LOAD (read from memory), STORE (write to memory), and EXECUTE (perform one of the internal 8087 math functions).

When the host executes a memory reference ESCAPE instruction intended to cause a read operation by the 8087, the host always reads the low-order word of any 8087 memory operand. The 8087 will save the address and data read. To read any subsequent words of the operand, the 8087 must become a local bus master.

When the 8087 has the local bus, it increments the 20-bit physical address it saved to address the remaining words of the operand.

When the ESCAPE instruction is intended to cause a write operation by the 8087, the 8087 will save the address but ignore the data read. Eventually, it will get control of the local bus and perform successive writes incrementing the 20-bit address after each word until the entire numeric variable has been written.

ESCAPE instructions intended to cause the execution of a coprocessor calculation do not require any bus activity. Numeric calculations work off of an internal register stack which has been initialized using a LOAD operation. The calculation takes place using one or two of the stack positions specified by the ESCAPE instruction. The result of the operation is also placed in one of the stack positions specified by the ESCAPE instruction. The result may then be returned to memory using a STORE instruction, thus allowing the host processor to access it.

4.0 OVERVIEW OF THE 82188 INTEGRATED BUS CONTROLLER

4.1 Introduction

The 82188 Integrated Bus Controller (IBC) is a highly integrated version of the 8288 Bus Controller. The IBC provides command and control timing signals for bus control and all of the necessary logic to interface the 80186 to the 8087.

4.2 Bus Control Signals

The bus command and control signals consist of \overline{RD} , \overline{WR} , \overline{DEN} , $\overline{DT/R}$, and \overline{ALE} . The timings and levels are driven following the latching of valid signals on the status lines $S0-S2$. When $S0-S2$ change state from passive to active, the IBC begins cycling through a state machine which drives the corresponding control and command lines for the bus cycle. As with the 8288, an address enable input (\overline{AEN}) is present to allow tri-stat-

ing when other bus masters supply their own bus control signals.

4.3 Bus Arbitration

The IBC also has the ability to convert bus arbitration protocols of $\overline{RQ}/\overline{GT}$ to \overline{HOLD} - \overline{HLDA} . This allows the 82586 Local Area Network (LAN) Coprocessor, the 82730 Text Coprocessor, and other coprocessors using the \overline{HOLD} - \overline{HLDA} protocol to be interfaced to the 8086/8 as well as allowing the 80186/8 to be interfaced to the 8087. In addition to converting arbitration protocols, the IBC makes it possible to arbitrate between two bus masters using \overline{HOLD} - \overline{HLDA} with a third using $\overline{RQ}/\overline{GT}$.

4.4 Interface Logic

In addition to all the bus control and arbitration features, the IBC provides logic to connect the queue status to the 8087, a chip-select for the 8087, and the necessary \overline{READY} synchronization required between the 8087 and the 80186/8.

5.0 DESIGNING THE SYSTEM

5.1 Circuit Schematics of the 80186/8-82888-8087 System

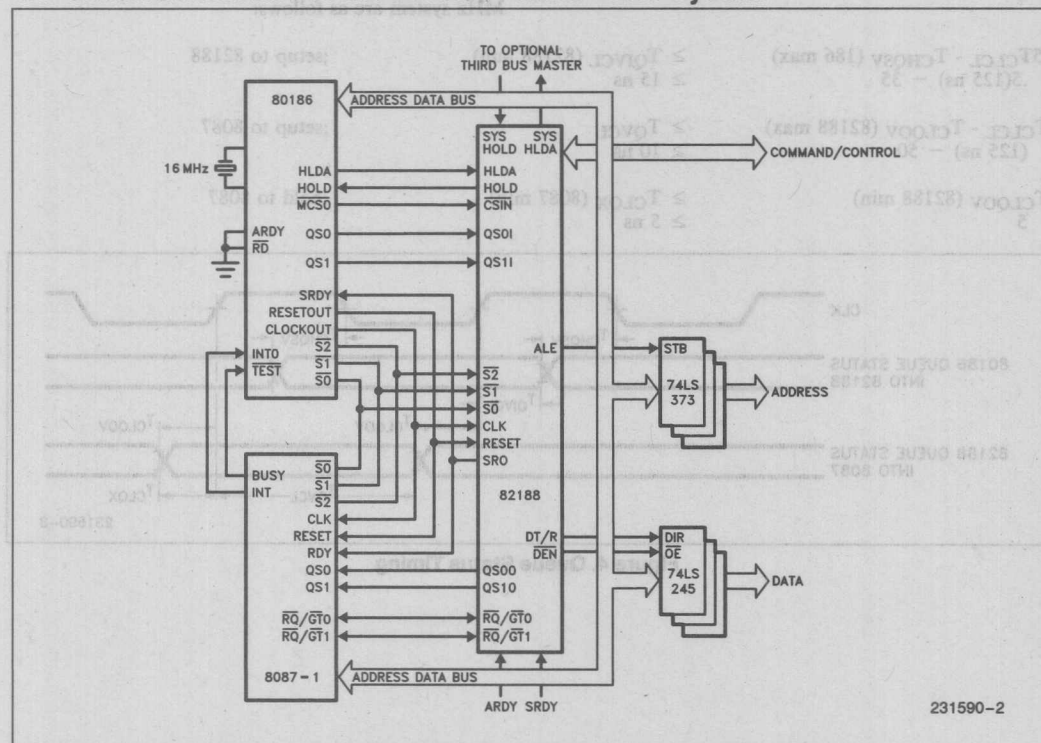


Figure 3. 80186/8-82188-8087 Circuit Diagram

5.2 Queue Status

The 8087 tracks the instruction execution of the 80186 by keeping an internal instruction queue which is identical to the processor's instruction queue. Each time the processor performs an instruction fetch, the 8087 latches the instruction into its own queue in parallel with the processor. Each time the processor removes the first byte of an instruction from the queue, the 8087 removes the byte at the top of the 8087 queue and checks to see if the byte is an ESCAPE prefix. If it is, the 8087 decodes the following bytes in parallel with the processor to determine which numeric instruction the bytes represent. If the first byte of the instruction is not an ESCAPE prefix, the 8087 discards it along with the subsequent bytes of the non-numeric instruction as the 80186 removes them from the queue for execution.

The 8087 operates its internal instruction queue by monitoring the two queue status lines from the CPU. This status information is made available by the CPU by placing it into queue status mode. This requires strapping the \overline{RD} pin on the 80186 to ground. When \overline{RD} is tied to ground, ALE and \overline{WR} become QS0 (Queue Status #0) and QS1 (Queue Status #1) respectively.

Table 1. Queue Status Decoding

QS1	QS0	Queue Operation
0	0	No queue operation
0	1	First byte from queue
1	0	Subsequent byte from queue
1	1	Reserved

Each time the 80186 begins decoding a new instruction, the queue status lines indicate "first byte of instruction taken from the queue". This signals the 8087 to check for an ESCAPE prefix. As the remaining bytes of the instruction are removed, the queue status indicates "subsequent byte removed from queue". The 8087 uses this status to either continue decoding subsequent bytes, if the first byte was an ESCAPE prefix, or to discard the subsequent bytes if the first byte was not an ESCAPE prefix.

The QS0(ALE) and QS1(\overline{WR}) pins of the 80186 are fed directly to the 82188 where they are latched and delayed by one-half-clock. The delayed queue status from the 82188 is then presented directly to the 8087.

The waveforms of the queue status signals are shown in Figure 4. The critical timings are the setup time into the 82188 from the 80186 and the setup and hold time into the 8087 from the 82188. The calculations for an 8 MHz system are as follows:

$$\begin{aligned}
 .5T_{CLCL} - T_{CHQSV} (186 \text{ max}) &\geq T_{QIVCL} (82188 \text{ min}) && \text{;setup to 82188} \\
 .5(125 \text{ ns}) - 35 &\geq 15 \text{ ns} \\
 T_{CLCL} - T_{CLQOV} (82188 \text{ max}) &\geq T_{QVCL} && \text{;setup to 8087} \\
 (125 \text{ ns}) - 50 &\geq 10 \text{ ns} \\
 T_{CLQOV} (82188 \text{ min}) &\geq T_{CLQX} (8087 \text{ min}) && \text{;hold to 8087} \\
 5 &\geq 5 \text{ ns}
 \end{aligned}$$

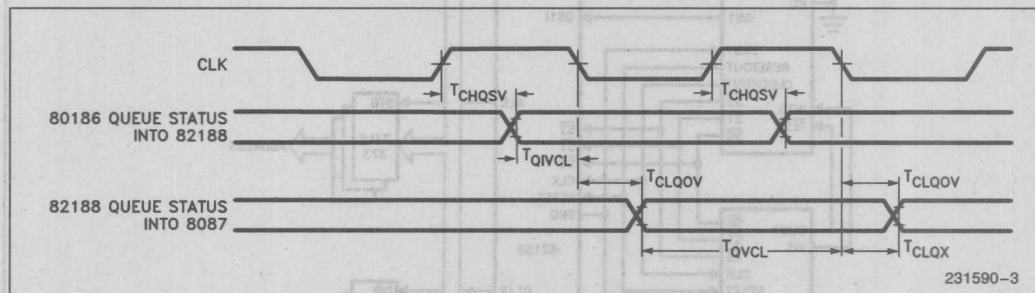


Figure 4. Queue Status Timing

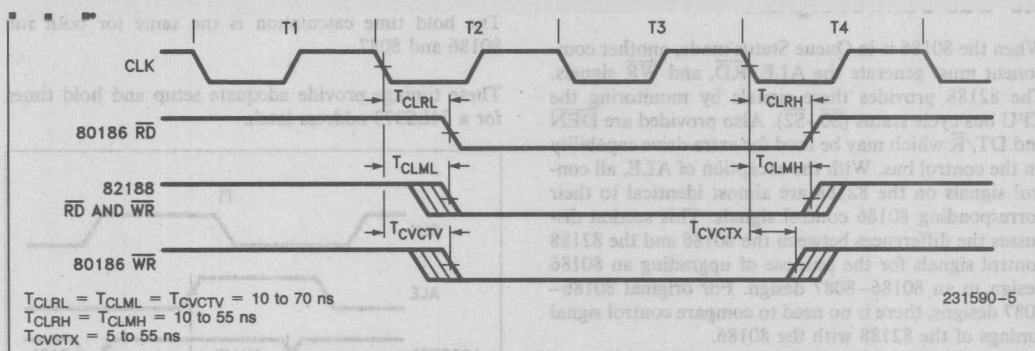


Figure 6. Read and Write Timings

5.3.2 Read and Write

The read and write signals of the 82188 have identical timings to those of the 80186 with one exception: the 82188 \overline{WR} inactive edge may not go inactive quite as early as the 80186. This spec is, in fact, a tighter spec than the 80186 \overline{WR} timing and should make designs easier. The timings for \overline{RD} and \overline{WR} are shown in Figure 6 for both the 80186 and the 82188.

5.3.3 \overline{DEN}

The \overline{DEN} signal on the 82188 is identical to the \overline{DEN} signal on the 80186 but with a tighter timing specification. This makes designs easier with the 82188 and makes upgrades from 80186 bus control to 82188 bus control more straightforward. The timings for \overline{DEN} on both the 80186 and 82188 are shown in Figure 7.

5.3.4 DT/\overline{R}

The operation of the DT/\overline{R} signal varies somewhat between the 80186 and the 82188. The 80186 DT/\overline{R} signal will remain in an active high state for all write cycles and will default to a high state when the system bus is idle (i.e., no bus activity). The 80186 DT/\overline{R} goes low only for read cycles and does so only for the duration of the bus cycle. At the end of the read cycle, assuming the following cycle is a non-read, the DT/\overline{R} signal will default back to a high state. Back-to-back read cycles will result in the DT/\overline{R} signal remaining low until the end of the last read cycle.

The DT/\overline{R} signal on the 82188 operates differently by making transitions only at the start of a bus cycle. The 82188 DT/\overline{R} signal has no default state and therefore will remain in whichever state the previous bus cycle required. The 82188 DT/\overline{R} signal will only change states when the current bus cycle requires a state different from the previous bus cycle.

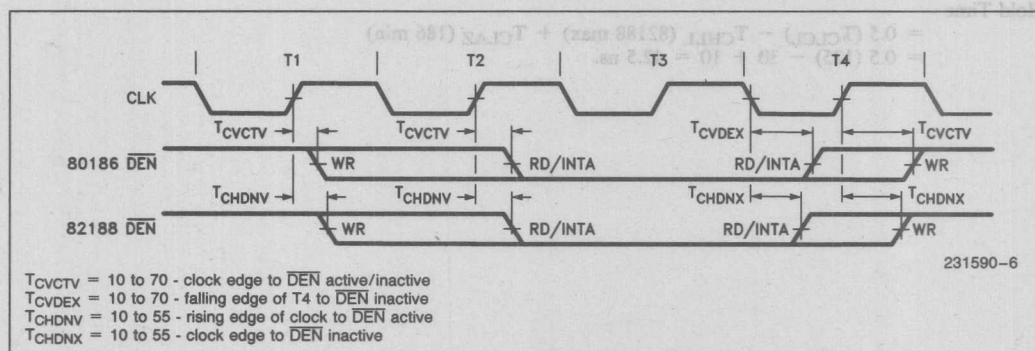


Figure 7. Data Control Timings

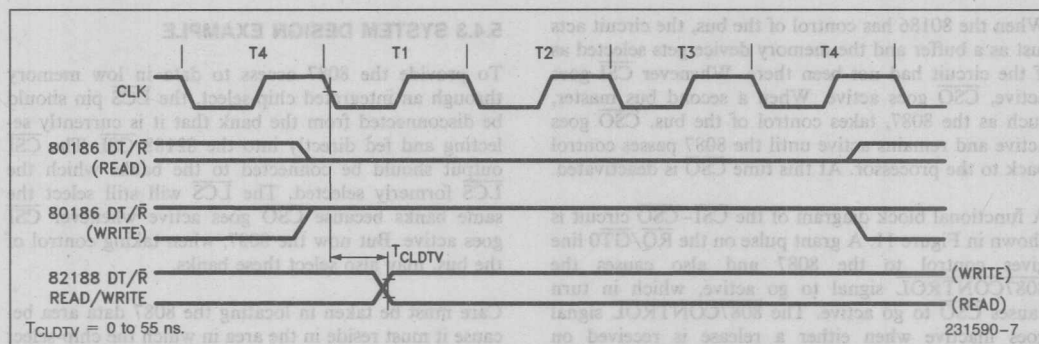


Figure 8. Data Transmit & Receive Timings

5.4 Chip Selects

5.4.1 INTRODUCTION

Chip-select circuitry is typically accomplished by using a discrete decoder to decode two or more of the upper address lines. When a valid address appears on the address bus, the decoder generates a valid chip-select. With this method, any bus master capable of placing an address on the system bus is able to generate a chip-select. An example of this is shown in Figure 9 where an 8086/8087 system uses a common decoder on the address bus. Note the decoder is able to operate regardless of which processor is in control of the bus.

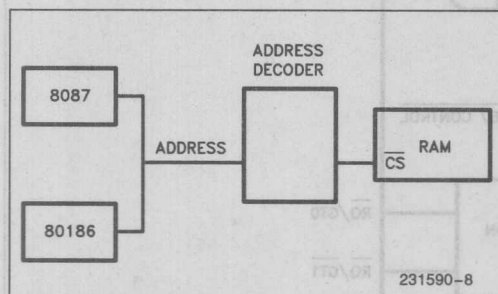


Figure 9. Typical 8086/8087 System

With high integration processors like the 80186 and 80188, the chip-select decoder is integrated onto the processor chip. The integrated chip-selects on the 80186 enable direct processor connection to the chip-enable pins on many memory devices, thus eliminating an external decoder. But because the integrated chip-selects decode the 80186's internal bus, an external bus master, such as the 8087, is unable to activate them. The 82188 IBC solves this problem by supplying a chip-select mechanism which may be activated by both the host processor and a second processor.

5.4.2 CSI and CSO OF THE 82188

The CSI (chip select in) and CSO (chip select out) pins of the 82188 provide a way for a second bus master to select memory while also making use of the 80186 integrated chip-selects. The CSI pin of the 82188 connects directly to one of the 80186's chip-selects while CSO connects to the memory device designated for the chip-selects range. An example of this is shown in Figure 10.

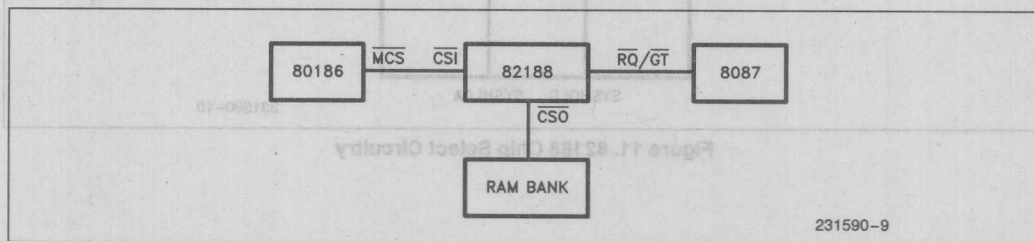


Figure 10. Typical 80186/82188/8087 System

When the 80186 has control of the bus, the circuit acts just as a buffer and the memory device gets selected as if the circuit had not been there. Whenever $\overline{\text{CSI}}$ goes active, $\overline{\text{CSO}}$ goes active. When a second bus master, such as the 8087, takes control of the bus, $\overline{\text{CSO}}$ goes active and remains active until the 8087 passes control back to the processor. At this time $\overline{\text{CSO}}$ is deactivated.

A functional block diagram of the $\overline{\text{CSI}}-\overline{\text{CSO}}$ circuit is shown in Figure 11. A grant pulse on the $\overline{\text{RQ/GT0}}$ line gives control to the 8087 and also causes the 8087CONTROL signal to go active, which in turn causes $\overline{\text{CSO}}$ to go active. The 8087CONTROL signal goes inactive when either a release is received on $\overline{\text{RQ/GT0}}$, indicating that the 8087 is relinquishing control to the main processor, or a grant is received on the $\overline{\text{RQ/GT1}}$ line, indicating that the 8087 is relinquishing control to a third processor. Both actions signify that the 8087 is relinquishing the bus. If $\overline{\text{CSO}}$ goes inactive because a third processor took control of the bus, then $\overline{\text{CSO}}$ will go active again for the 8087 when a release pulse is transmitted on the $\overline{\text{RQ/GT1}}$ line to the 8087. This release pulse occurs as a result of SYSHLDA going inactive from the third processor.

5.4.3 SYSTEM DESIGN EXAMPLE

To provide the 8087 access to data in low memory through an integrated chip-select, the $\overline{\text{LCS}}$ pin should be disconnected from the bank that it is currently selecting and fed directly into the 82188 $\overline{\text{CSI}}$. The $\overline{\text{CSI}}$ output should be connected to the banks which the $\overline{\text{LCS}}$ formerly selected. The $\overline{\text{LCS}}$ will still select the same banks because $\overline{\text{CSO}}$ goes active whenever $\overline{\text{CSI}}$ goes active. But now the 8087, when taking control of the bus, may also select these banks.

Care must be taken in locating the 8087 data area because it must reside in the area in which the chip-select is defined. If the 8087 generates an address outside of the $\overline{\text{LCS}}$ range, the $\overline{\text{CSO}}$ will still go active, but the address will erroneously select a part of the lower bank. Note also that this chip-select limits the size of the 8087 data area to the maximum size memory which can be selected with one chip-select. However, this does not place a limit on instruction code size or non-8087 data size. All 80186 and 8087 instructions are fetched by the processor and therefore do not require that the 8087 be

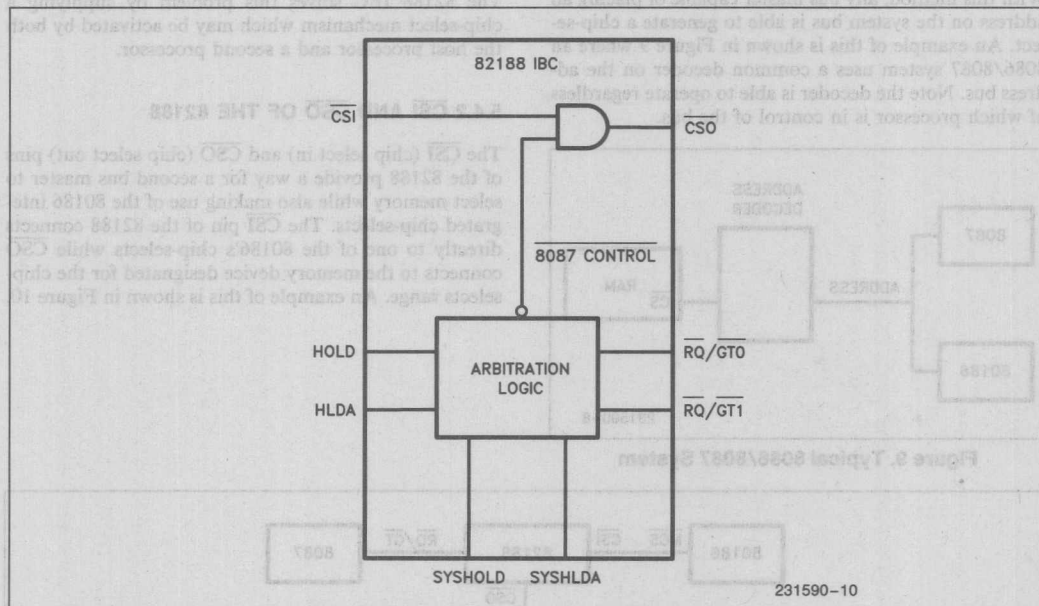


Figure 11. 82188 Chip Select Circuitry

able to address them. Likewise, non-8087 data is never accessed by the 8087 and therefore does not require an 8087 chip-select.

5.5 Wait State & Ready Logic

The 8087 must accurately track every instruction fetch the 80186 performs so that each op-code may be read from the system bus by the 8087 in parallel with the processor. This means that for instruction code areas, the 80186 cannot use internally generated wait states. All ready logic for these areas must be generated externally and sent into the 82188. The 82188 then presents a synchronous ready out (SRO) signal to both the 80186 and the 8087.

5.5.1 INTERNAL WAIT STATES WITH INSTRUCTION FETCHES

If internal wait states are used by the processor with the 8087 at zero wait states, then the 8087 will latch op-codes using a four clock bus cycle while the processor is using between five and seven clocks on each bus cycle. If the wait states are truly necessary to latch valid data from memory, then a four clock bus cycle will force the 8087 to latch invalid data. The invalid data may then be possibly interpreted to be an ESCAPE prefix when, in reality, it is not. The reverse may also hold true in that the 8087 may not recognize an ESCAPE prefix when it is fetched. These conditions could cause a system to hang (i.e., cease to operate), or operate with erroneous results.

If the memory is fast enough to allow latching of valid data within a four clock bus cycle, then the 80186 internal wait states will not cause the system to hang. Both processors will receive valid data during their respective bus cycles. The 8087 will finish its bus cycle earlier than the processor, but this is of no consequence to system operation. The 8087 will synchronize with the processor using the status lines S0-S2 at the start of the next instruction fetch.

5.5.2 INTERNAL WAIT STATES WITH DATA & I/O CYCLES

With the exception of "Dummy Read Cycles" and instruction fetches, all memory and I/O bus cycles executed by the host processor are ignored by the 8087. Coprocessor synchronization is not required for untracked bus cycles and, therefore, internally generated wait states do not affect system operation. All of the I/O space and any part of memory used strictly for data may use the internal wait state generator on the 80186.

Memory used for 8087 data is somewhat different. Here, as in the case of code segment areas, the system must rely on an external ready signal or else the memory must be fast enough to support zero wait state operation. Without an external ready signal, the 8087 will always perform a four clock bus cycle which, when used with slow memories, results in the latching of invalid data.

Internal wait states will not affect system operation for data cycles performed by the 8087. In this case the 8087 has control of the bus and the two processors operate independently.

One type of data cycle has not yet been considered. Each time a numeric variable is accessed, the host processor runs a "Dummy Read Cycle" in order to calculate the operand address for the 8087. The 8087 latches the address and then takes control of the bus to fetch any subsequent bytes which are necessary. If the 8087 variables are located at even addresses, then an internally generated wait state will not present any problems to the system. If any numeric variables are located at odd addresses, then the interface between the host and coprocessor becomes asynchronous causing erroneous results.

The erroneous results are due to the 80186 running two back-to-back bus cycles with wait states while the 8087 runs two back-to-back bus cycles without wait states. The start of the second bus cycle is completely uncoordinated between the two processors and the 8087 is unable to latch the correct address for subsequent transfers. For this reason, 8087 variables in a 80186 system must always lie on even boundaries when using the internal wait state generator to access them.

Numeric variables in an 80188 system must never be in a section of memory which uses the internal wait state generator. The 80188 will always perform consecutive bus cycles which would be equivalent to the 80186 performing an odd addressed "Dummy Read Cycle."

5.5.3 AUTOMATIC WAIT STATES AT RESET

The 80186 automatically inserts three wait states to the predefined upper memory chip select range upon power up and reset. This enables designers to use slow memories for system boot ROM if so desired. If slow ROM's are chosen, then no further programming is necessary. If fast ROM's are chosen, then the wait state logic may simply be reprogrammed to the appropriate number of wait states.

The automatic wait states have the possibility of presenting the same problem as described in section 5.5.1 if

the boot ROM needs one or more wait states. Under these conditions the 8087 would be forced to latch invalid opcodes and possibly mistake one for an ESCAPE instruction.

If the boot ROM requires wait states, then some sort of external ready logic is necessary. This allows both processors to run with the same number of wait states and insures that they always receive valid data.

If the boot ROM does not require wait states, then there is no need to design external ready logic for the upper chip select region. But if 8087 code is present in the upper memory chip select region, the situation described in section 3.4 regarding "Dummy Read Cycles" must be considered.

The 82188 solves this problem by inserting three wait states on the SRO line to the 8087 for the first 256 bus cycles. By doing this the 82188 inserts the same number of wait states to both processors keeping them synchronized. The initialization code for the 80186 must program the upper memory chip select to look at external ready and to insert zero wait states within these first 256 bus cycles. At the end of the 256 bus cycles, the 82188 stops inserting wait states and both processors run at zero wait states.

5.5.4 EXTERNAL READY SYNCHRONIZATION

The 80186 and 8087 sample READY on different clock edges. This implies that some sort of external synchronization is required to insure that both processors sample the same ready state. Without the synchronization, it would be possible for the external signal to change state between samples. The 80186 may sample ready high while the 8087 samples ready low. This would lead to the two processors running different length bus cycles and possibly cause the system to hang.

The 82188 provides ready synchronization through the ARDY and SRDY inputs. Once a valid transition is recorded, the 82188 presents the results on the SRO output and holds the output in that state until both processors have had a chance to sample the signal.

5.6 BUS ARBITRATION

In order for the 8087 to read and write numeric data to and from memory, it must have a means of taking control of the local bus. With the 8086/88 this is accomplished through a request-grant exchange protocol. The 80186, however, makes use of HOLD/HOLD AC-

KNOWLEDGE protocol to exchange control of the bus with another processor. The 82188 supplies the necessary conversion to interface $\overline{RQ/GT}$ to HOLD/HLDA signals. The $\overline{RQ/GT}$ signal of the 8087 connects directly to the 82188's $\overline{RQ/GT0}$ input while the 82188's HOLD and HLDA pins connect to the 80186's HOLD and HLDA pins.

When the 8087 requires control of the bus, the 8087 sends a request on the $\overline{RQ/GT0}$ line to the 82188. The 82188 responds by sending a HOLD request to the 80186. When HLDA is received back from the 80186, the 82188 sends a grant back to the 8087 on the same $\overline{RQ/GT0}$ line.

The 82188 also has provisions for adding a third bus-master to the system which uses HOLD/HLDA protocol. This is accomplished by using the 82188 SYSHOLD, SYSHLDA, and $\overline{RQ/GT1}$ signals. The third processor requests the bus by pulling the SYSHOLD line high. The 82188 will route (and translate if necessary) the requests to the current bus master. If the 8087 has control, the 82188 will request control via the $\overline{RQ/GT1}$ line which should be connected to the 8087's $\overline{RQ/GT1}$ line.

The 8087 will relinquish control by getting off the bus and sending a grant pulse on the $\overline{RQ/GT1}$ line. The 82188 responds by sending a SYSHLDA to the third processor. The third processor lowers SYSHOLD when it has finished on the bus. The 82188 routes this in the form of a release pulse on the $\overline{RQ/GT1}$ line to the 8087. The 8087 then continues bus activity where it left off. The maximum latency from SYSHOLD to SYSHLDA is equal to the 80186 latency + 8087 latency + 82188 latency.

5.7 SPEED REQUIREMENTS

One of the most important timing specs associated with the 80186-8087 interface is the speed at which the system should run. The 8087 was designed to operate with a 33% duty cycle clock whereas the 80186 and 80188 were designed to operate with a 50% duty cycle clock. In order to run both parts off the same clock, the 8087 must run at a slower speed than is typically implied by its dash number in the 8086/88 family.

To determine the speed at which an 8087 may run (with a 50% duty cycle clock), the minimum low and high times of the 8087 must be examined. The maximum of these two minimum specs becomes the half-period of the 50% duty cycle system clock. For example, the 8087-1 provides worst case spec compatibility with the 80186 at system clock-speeds of up to 8 MHz. The clock waveforms are shown in Figure 12 using 10 MHz timings.

The minimum clock low time spec (T_{CLCH}) of the 10 MHz 8087 is 53 ns. The clock low time of an 8 MHz 80186 is specified to be:

$$\frac{1}{2}(T_{CLCL}) - 7.5$$

Solving for T_{CLCL} of the 80186 using T_{CLCH} of the 8087 yields the following:

$$\frac{1}{2}(T_{CLCL}) - 7.5 = T_{CLCH}$$

$$(T_{CLCL}) = 2(T_{CLCH} + 7.5)$$

$$T_{CLCL} = 121 \text{ ns}$$

The calculation shows minimum cycle time of the 80186 to be 121 ns. This time translates into a maximum frequency of 8.26 MHz.

6.0 BENCHMARKS

6.1 Introduction

The following benchmarks compare the overall system performance of an 8086, 80188, and an 80186 in numeric applications. Results are shown for all three processors in systems with the 8087 coprocessor and in systems using an 8087 software emulator. Three FORTRAN benchmark programs are used to dem-

onstrate the large increase in floating-point math performance provided by the 8087 and also the increase in performance due to the enhanced 80186 and 80188 host processors.

The 8086 results were measured on an Intel® Series III Microcomputer Development System with an iSBC® 86/12 board and an iSBC 337 multimodule. Typically, one wait state for memory read cycles and two wait states for memory write cycles are experienced in this environment.

The 80186 and 80188 results were measured on a prototype board which allowed zero wait state operation at 8 MHz. The benchmarks measured using the 8087 showed little sensitivity to wait states. Instructions executed on the 8087 tend to be long in comparison to the amount of bus activity required and, therefore, are not affected much by wait states.

The benchmarks measured using the software emulator are much more bus intensive and average from 10 to 15 percent performance degradation for one wait state.

All execution times shown here represent 8 MHz operation. The 8086 results were measured at 5 MHz and extrapolated to achieve 8 MHz execution times.

6.2 Interest Rate Calculations

Routines were written in FORTRAN-86 to calculate the final value of a fund given the annual interest and the present value. It is assumed that the interest will be compounded daily, which requires the calculation of the yearly effective rate. This value, which is the equivalent annual interest if the interest were compounded daily, is determined by the following formula:

$$\text{year} = (1 + (ir/np))^{np} - 1$$

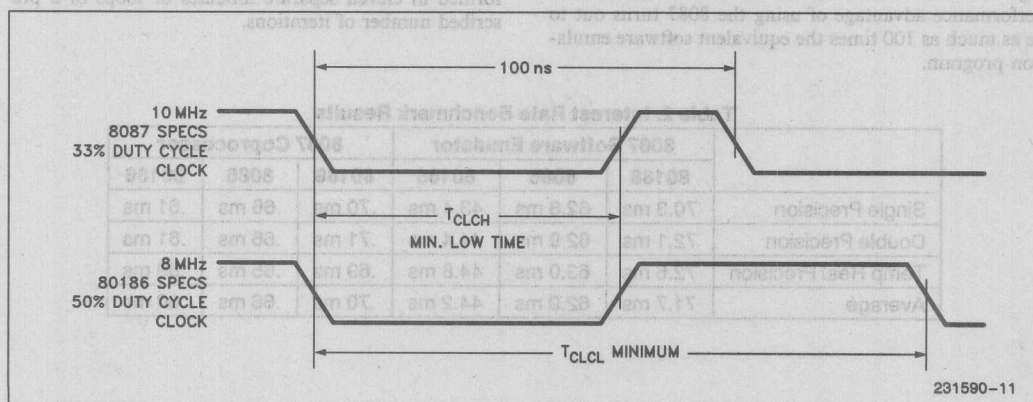


Figure 12. Clock Cycle Timing

where:
 yer is the yearly effective rate
 ir is the annual interest rate
 np is the number of compounding periods per annum

Once the yer is determined, the final value of the fund is determined by the formula:

$$fv = (1 + yer) * pv$$

where:
 pv is the present value
 fv is the future value

Results are obtained using single-precision, double-precision, and temporary real precision operands when:

ir is set to 10% (0.1)
 np is set to 365 (for daily compounding)
 pv is set to \$2,000,000

THE RESULTS:

	yer	Final Value
Single-Precision (32-bit)	10.514%	\$2,210,287.50
Double-Precision (64-bit)	10.516%	\$2,210,311.57
Temporary Real Precision	10.516%	\$2,210,311.57

The difference between the final single-precision and double-precision values is \$24.07; the difference in the final value between the double-precision and the temporary real precision is 0.000062 cents. Since the 8087 performs all internal calculations on 80-bit floating point numbers (temp real format), temporary real precision operations perform faster than single- or double-precision. No data conversions are required when loading or storing temporary real values in the 8087. Thus, for business applications, the double-precision computing of the 8087 is essential for accurate results, and the performance advantage of using the 8087 turns out to be as much as 100 times the equivalent software emulation program.

6.3 Matrix Multiply Benchmark Routine

A routine was written in FORTRAN-86 to compute the product of two matrices using a simple row/column inner-product method. Execution times were obtained for the multiplication of 32×32 matrices using double precision. The results of the benchmark are shown in Figure 14.

The results show the 8087 coprocessor systems performing from 23 to 31 times faster than the equivalent software emulation program. Both the 80188/87 and the 80186/87 systems outperform the 8086/87 system by 34 to 75 percent. This difference is mainly attributed to the fact that the matrix program largely consists of effective address calculations used in array accessing. The hardware effective address calculator of the 80186 and 80188 allow each array access to improve by as much as three times the 8086 effective address calculation.

6.4 Whetstone Benchmark Routine

The Whetstone benchmark program was developed by Harry Curnow for the Central Computer Agency of the British government. This benchmark has received high visibility in the scientific community as a measurement of main frame computer performance. It is a "synthetic" program. That is, it does not solve a real problem, but rather contains a mix of FORTRAN statements which reflect the frequency of such statements as measured in over 900 actual programs. The program computes a performance metric: "thousands of Whetstone instructions per second (KIPS)."

Simple variable and array addressing, fixed- and floating-point arithmetic, subroutine calls and parameter passing, and standard mathematical functions are performed in eleven separate modules or loops of a prescribed number of iterations.

Table 2. Interest Rate Benchmark Results

	8087 Software Emulator			8087 Coprocessor		
	80188	8086	80186	80188	8086	80186
Single Precision	70.3 ms	62.8 ms	43.4 ms	.70 ms	.66 ms	.61 ms
Double Precision	72.1 ms	62.9 ms	44.4 ms	.71 ms	.66 ms	.61 ms
Temp Real Precision	72.6 ms	63.0 ms	44.8 ms	.69 ms	.65 ms	.59 ms
Average	71.7 ms	62.9 ms	44.2 ms	.70 ms	.66 ms	.60 ms

The original coding of the Whetstone benchmark was written in Algol-60 and used single-precision values. It was rewritten in FORTRAN with single-precision values to exactly reflect the original intent. Another version was created using double-precision values. The results are shown in Table 3.

The results show the 8087 systems with the 80186 and 80188 outperforming the equivalent software emulation by 60 to 83 times. Additionally, the 80186 coupled with the 8087 outperformed the 8086/87 system by 22 percent.

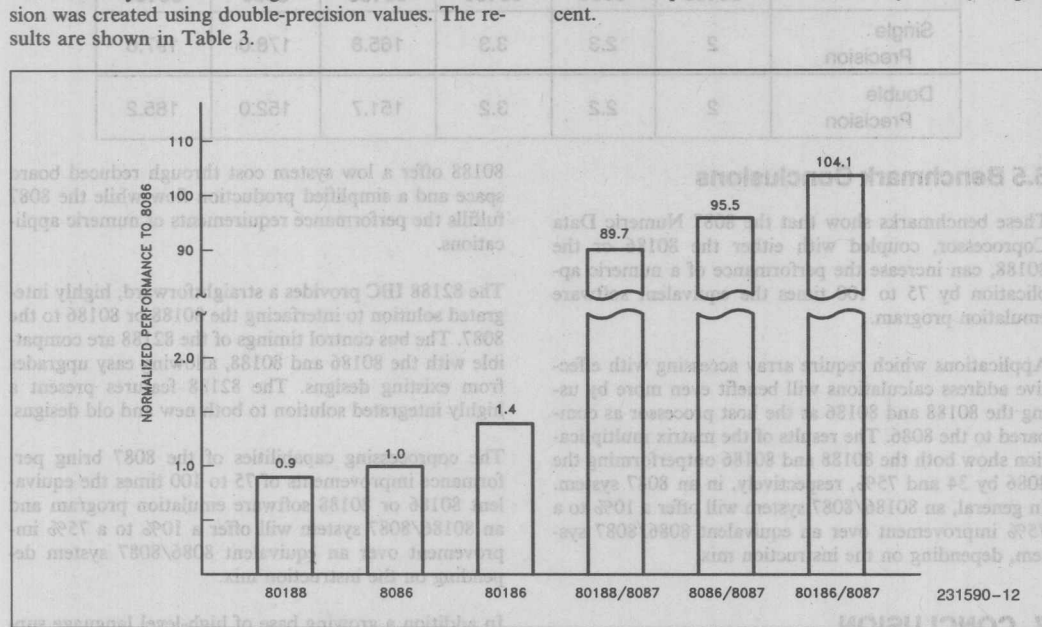


Figure 13. Interest Rate Benchmark Results

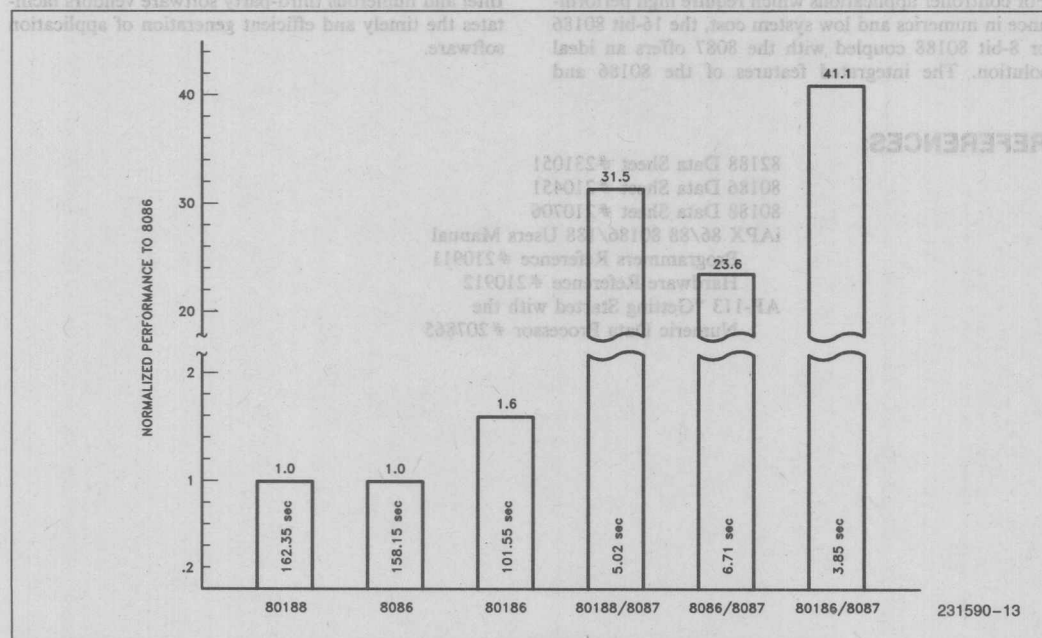


Figure 14. Double Precision Matrix Multiplication

Units = KIPS	8087 Software Emulator			8087 Coprocessor		
	80188	8086	80186	80188	8086	80186
Single Precision	2	2.3	3.3	165.8	178.0	197.6
Double Precision	2	2.2	3.2	151.7	152.0	185.2

6.5 Benchmark Conclusions

These benchmarks show that the 8087 Numeric Data Coprocessor, coupled with either the 80186 or the 80188, can increase the performance of a numeric application by 75 to 100 times the equivalent software emulation program.

Applications which require array accessing with effective address calculations will benefit even more by using the 80188 and 80186 as the host processor as compared to the 8086. The results of the matrix multiplication show both the 80188 and 80186 outperforming the 8086 by 34 and 75%, respectively, in an 8087 system. In general, an 80186/8087 system will offer a 10% to a 75% improvement over an equivalent 8086/8087 system, depending on the instruction mix.

80188 offer a low system cost through reduced board space and a simplified production flow while the 8087 fulfills the performance requirements of numeric applications.

The 82188 IBC provides a straightforward, highly integrated solution to interfacing the 80188 or 80186 to the 8087. The bus control timings of the 82188 are compatible with the 80186 and 80188, allowing easy upgrades from existing designs. The 82188 features present a highly integrated solution to both new and old designs.

The coprocessing capabilities of the 8087 bring performance improvements of 75 to 100 times the equivalent 80186 or 80188 software emulation program and an 80186/8087 system will offer a 10% to a 75% improvement over an equivalent 8086/8087 system depending on the instruction mix.

7. CONCLUSION

For controller applications which require high performance in numerics and low system cost, the 16-bit 80186 or 8-bit 80188 coupled with the 8087 offers an ideal solution. The integrated features of the 80186 and

In addition a growing base of high-level language support (FORTRAN, Pascal, C, Basic, PL/M, etc.) from Intel and numerous third-party software vendors facilitates the timely and efficient generation of application software.

REFERENCES:

- 82188 Data Sheet #231051
- 80186 Data Sheet #210451
- 80188 Data Sheet #210706
- iAPX 86/88 80186/188 Users Manual
- Programmers Reference #210911
- Hardware Reference #210912
- AP-113 "Getting Started with the
- Numeric Data Processor #207865

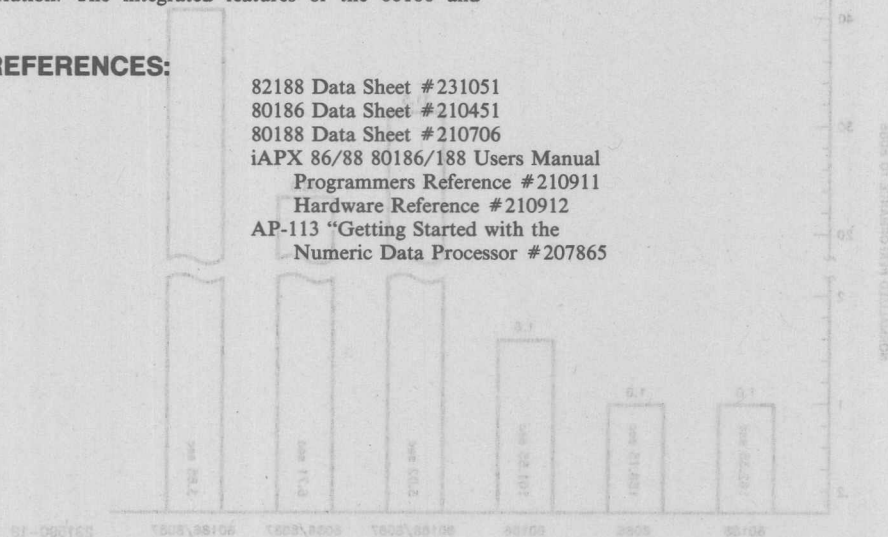


Figure 14: Double Precision Matrix Multiplication



APPLICATION NOTE

AP-286

October 1986

80186/188 Interface to Intel Microcontrollers

5

PARVIZ KHODADADI
APPLICATIONS ENGINEER

Order Number: 231784-001

80186/188 INTERFACE TO INTEL MICROCONTROLLERS

CONTENTS

PAGE

1.0 INTRODUCTION	5-22
1.1 System Overview	5-23
1.2 Application Examples	5-23
2.0 OVERVIEW OF THE 80186, 80C51, 8052, AND 8044	5-23
2.1 The 80186 Internal Architecture	5-23
2.2 The MCS-51 Internal Architecture	5-24
2.3 The 8044 Internal Architecture	5-24
3.0 80186/MICROCONTROLLER INTERACTION	5-25
4.0 SYSTEM INTERFACE	5-26
4.1 Command/Status Transfers	5-26
4.2 Data/Parameter Transfer	5-26
4.3 Interrupts	5-27
5.0 COMMAND AND STATUS	5-28
5.1 Commands	5-28
5.1.1 Acknowledging Interrupt	5-28
5.1.2 Operations	5-28
5.1.3 Illegal Commands	5-30
5.2 Status	5-30
5.2.1 Interrupt	5-30
5.2.2 DMA Operation	5-30
5.2.3 Error	5-30
5.2.4 Request to Send	5-30
5.2.5 Clear to Send	5-30
5.2.6 Event	5-30
6.0 HARDWARE DESCRIPTION	5-31
6.1 Reset	5-31
6.2 Sending Commands	5-31
6.3 DMA Transfers	5-32
6.4 Reading Status	5-32

CONTENTS	PAGE
7.0 80186/8044 INTERFACE	5-33
7.1 Configuring the 8044	5-33
7.2 Transferring a Message with the 8044	5-34
7.3 Receiving a Message with the 8044	5-34
7.4 Dumping the 8044 Registers	5-35
7.5 Aborting an Operation	5-35
7.6 Disabling Transmission or Reception	5-35
7.7 Handling Interrupts	5-36

Identical to the 8044, 80C21, and the 80C22 is identical because they have identical pinouts (some pins have alternate functions). As an example, the software provided for the 8044/80186 interface, which is the building block for the application driver, is applied in this Application Note.

CONTENTS	PAGE
8.0 8044 IN EXPANDED OPERATION	5-36
8.1 Transmitting a Message in Expanded Operation	5-36
8.2 Receiving a Message in Expanded Operation	5-36
9.0 CONCLUSION	5-36

APPENDIX A: SOFTWARE	5-37
-----------------------------	------

to increase performance and achieve flexibility. One alternative solution is the combination of the Intel highly integrated 80186 microprocessor and the Intel 8-bit microcontroller such as the 80C21, 80C22, or 8044. In such a system, the 80186 provides the processing power and the 1 Mbyte memory addressability, while the microcontroller provides the intelligence for the I/O operations and data communication tasks. The 80186 runs application programs, performs the high level communication tasks, and provides the human interface. The microcontroller performs 8-bit math and single bit logic operations, the low level communication tasks, and I/O processing.

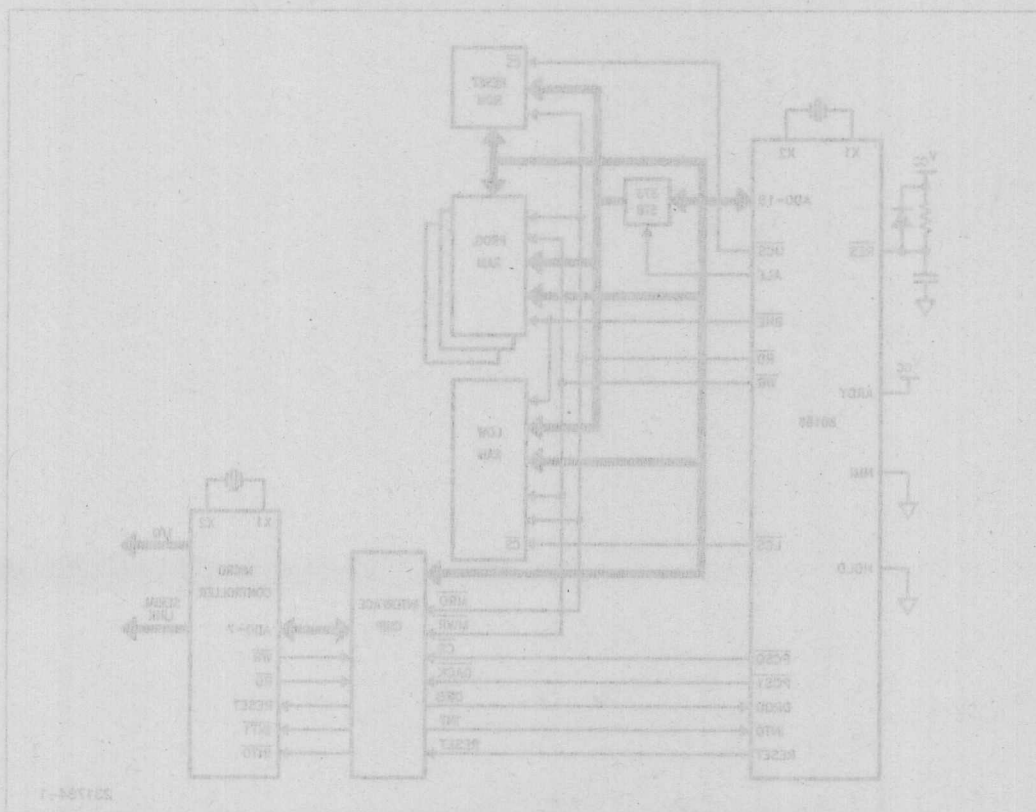


Figure 7.1: 80186/Microcontroller Based System

1.0 INTRODUCTION

Systems which require I/O processing and serial data transmission are very software intensive. The communication task and I/O operations consume a lot of the system's intelligence and software. In many conventional systems the central processing unit carries the burden of all the communication and I/O operations in addition to its main routines, resulting in a slow and inefficient system.

In an ideal system, tasks are divided among processors to increase performance and achieve flexibility. One attractive solution is the combination of the Intel highly integrated 80186 microprocessor and the Intel 8-bit microcontrollers such as the 80C51, 8052, or 8044. In such a system, the 80186 provides the processing power and the 1 Mbyte memory addressability, while the controller provides the intelligence for the I/O operations and data communication tasks. The 80186 runs application programs, performs the high level communication tasks, and provides the human interface. The microcontroller performs 8-bit math and single bit boolean operations, the low level communication tasks, and I/O processing.

This application note describes an efficient method of interfacing the 16-bit 80186 high integration microprocessor to the 80C51, 8052, or the microcontroller-based 8044 serial communication controller. The interface hardware shown in Figure 1.1, is very simple and may be implemented with a programmable logic device or a gate-array. The 80186 and the microcontroller may run asynchronously and at different speeds. With this technique data transfers up to 200 Kbytes per second can be achieved between a 12 MHz microcontroller and an 8 MHz 80186.

The 8-bit 80188 high integration microprocessor can also be used with the same interface technique. The performance of the interface is the same since an 8-bit bus is used.

Interface to the 8044, 80C51, and the 8052 is identical because they have identical pinouts (some pins have alternate functions). As an example, the software procedures for the 8044/80186 interface, which is the building block for the application driver, is supplied in this Application Note.

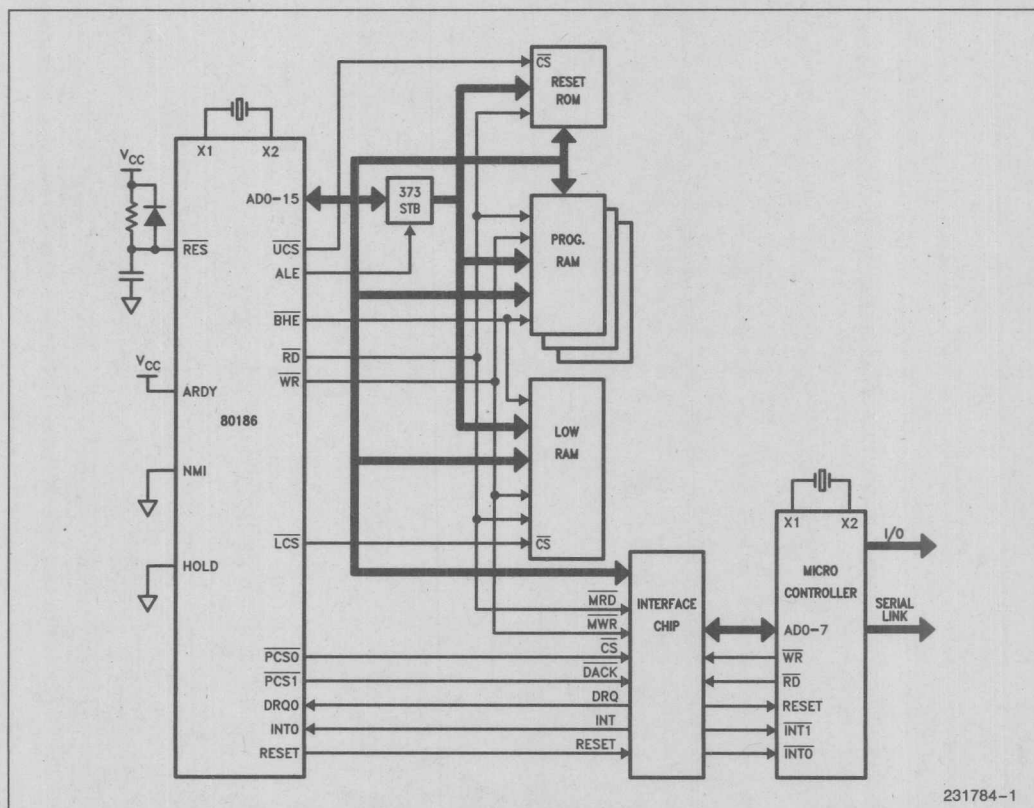


Figure 1.1. 80186/Microcontroller Based System

1.1 System Overview

The 80186 and the microcontrollers are processors. They each access memory and have address/data, read, and write signals. There are three common ways to interface multiple processors together:

- 1) First In First Out (FIFO)
- 2) Dual Port RAM (DPRAM)
- 3) Slave Port

The FIFO interface, compared to DPRAM, requires less TTL and is easier to interface; however, FIFOs are expensive. The DPRAM interface is also expensive and even more complex. When DPRAM is used, the address/data lines of each processor must be buffered, and hardware logic is needed to arbitrate access to DPRAM. The slave port interface given here is cheaper and easier than both FIFO and DPRAM alternatives.

The 80186 processor, when interfaced to this circuit, views the microcontroller as a peripheral chip with 8-bit data bus and no address lines (see Figure 1.1). It can read status and send commands to the microcontroller at any time. The microcontroller becomes a slave co-processor while keeping its processing power and serial communication capabilities.

The microcontrollers, with the interface hardware, have a high level command interface like many other data communication peripherals. For example, the 80186 can send the microcontroller commands such as Transmit or Configure. This means the designer does not have to write low level software to perform these tasks, and it offloads the 80186 to serve other functions in the application.

1.2 Application Examples:

The combination of the 80186 and a microcontroller basically provides all the functions that are needed in a system: a 16-bit CPU, 8-bit CPU, DMA controller, I/O ports, and a serial port. The 80C51 and the 8052 have an on-chip asynchronous channel, while the 8044 has an intelligent SDLC serial channel. In addition, many other functions such as timers, counters, and interrupt controllers are integrated in both the 80186 and the microcontrollers.

Applications of the system described above are in the area of robotics, data communication networks, or serial communication backplanes. A typical example is copiers. Different segments of the copy machine like the motor, paper feed, diagnostics, and error/warning displays are all controlled by microcontrollers. Each segment receives orders from and replies to the central processor which consists of the 80186 interfaced with a microcontroller.

Another common application is in the area of process controllers. An example is a central control unit for a multiple story building which controls the heating, cooling, and lighting of each room in each floor. In each room a microcontroller performs the above functions based on the orders received from the central processor. Depending on the throughput and type of the serial communication required, the 8044 or the 80C51 (8052) may be selected for the application.

2.0 OVERVIEW OF THE 80186, 80C51, 8052, AND 8044

This section briefly discusses the features of the microcontrollers and the 80186. For more information about these products please refer to the Intel Microcontroller and Microsystem components hand-books. Readers familiar with the above products may skip this section.

2.1 The 80186 Internal Architecture

The 80186 contains an enhanced version of Intel's popular 8086 CPU integrated with many other features common to most systems (Figure 2.1). The 16-bit CPU can access up to 1 Mbyte of memory and execute instructions faster than the 8086. With speed selection of 8, 10, and 12.5 MHz, this highly integrated product is the most popular 16-bit microprocessor for embedded control applications.

The on-chip DMA controller has two channels which can each be shared by multiple devices. Each channel is capable of transferring data up to 3.12 Mbytes per second (12.5 MHz speed). It offers the choice of byte or word transfer. It can be programmed to perform a burst transfer of a block of data, transfer data per specified time interval, or transfer data per external request.

The on-chip interrupt controller responds to both external interrupts and interrupts requested by the on-chip peripherals such as the timers and the DMA channels. It can be configured to generate interrupt vector addresses internally like the microcontrollers or externally like the popular 8259 interrupt controller. It can be configured to be a slave controller to an external interrupt controller (iRMX 86 mode) or be master for one or two 8259s which in turn may be masters for up to 8 more 8259s. When configured in master mode, each channel can support up to 64 external interrupts (128 total).

Three 16-bit timers are also integrated on the chip. Timer 0 and timer 1 can be configured to be 16-bit counters and count external events. If configured as timers, they can be started by software or by an external event. Timer 0 and 1 each contain a timer output pin. Transitions on these pins occur when the timers reach one of the two possible maximum counts. Timer

2 can be used as a prescaler for timer 0 and 1 or can be used to generate DMA requests to the on-chip DMA channel.

Finally, the integrated clock generator, the wait state generator, and the chip select logic reduce the external logic necessary to build a processing system.

2.2 The MCS-51 Internal Architecture

The 80C51BH, as shown in Figure 2.2, consists of an 8-bit CPU which can access up to 64 Kbytes of data memory (RAM) and 64 Kbytes of program memory (ROM). In addition, 4 Kbytes of ROM and 128 bytes of RAM are built onto the chip.

The on-chip interrupt controller supports five interrupts with two priority levels. There are two timers integrated in the 80C51. Timer 0 and 1 can be configured as 8-bit or 16-bit timers or event counters.

Finally the integrated full duplex asynchronous serial channel provides the human interface or communica-

tion capability with other microcontrollers. The UART supports data rates up to 500 kHz (with 15 MHz crystal) and can distinguish between address bytes and data bytes.

The 8052 has the same features as the 80C51 except it has 8 Kbytes of on-chip ROM and 256 bytes of on-chip RAM. In addition the 8052 has another timer which may be configured as the baud rate generator for the serial port.

2.3 The 8044 Internal Architecture

The 8044 has all the features of the 80C51. In addition the on-chip RAM size is increased to 192 bytes and an intelligent HDLC/SDLC serial channel (SIU) replaces the 80C51 serial port (see Figure 2.3). It supports data rates up to 2.4 Mbps when an external clock is used and 375 Kbps when the clock is extracted from the data line. The serial port can be used in half duplex point to point, multipoint, or one-way loop configurations.

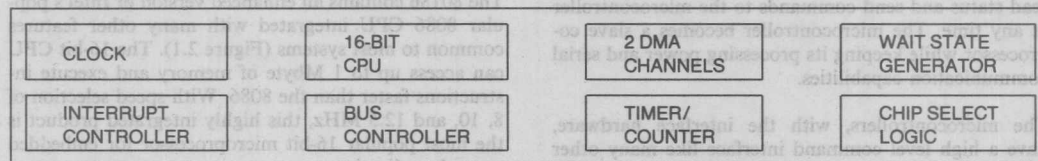


Figure 2.1. 80186 Block Diagram

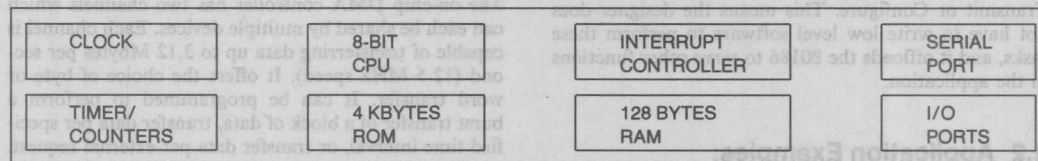


Figure 2.2. 80C51 Block Diagram

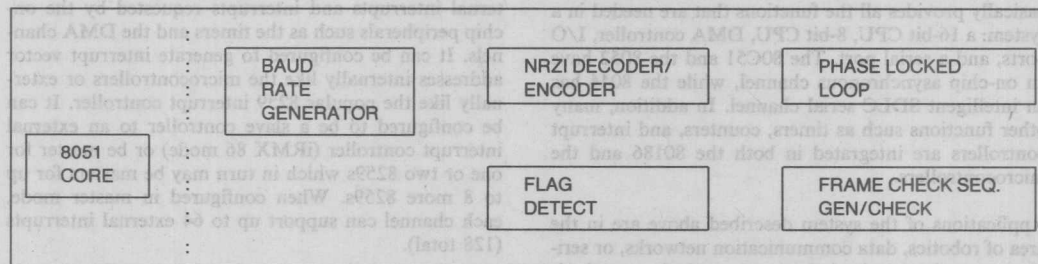


Figure 2.3. 8044 Block Diagram

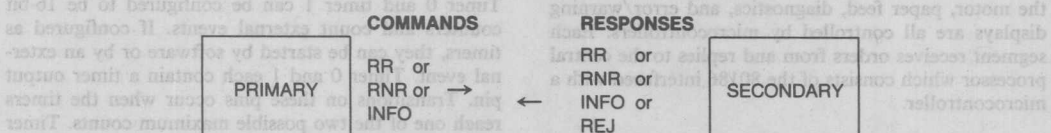


Figure 2.4. 8044 Automatic Response to SDLC Commands

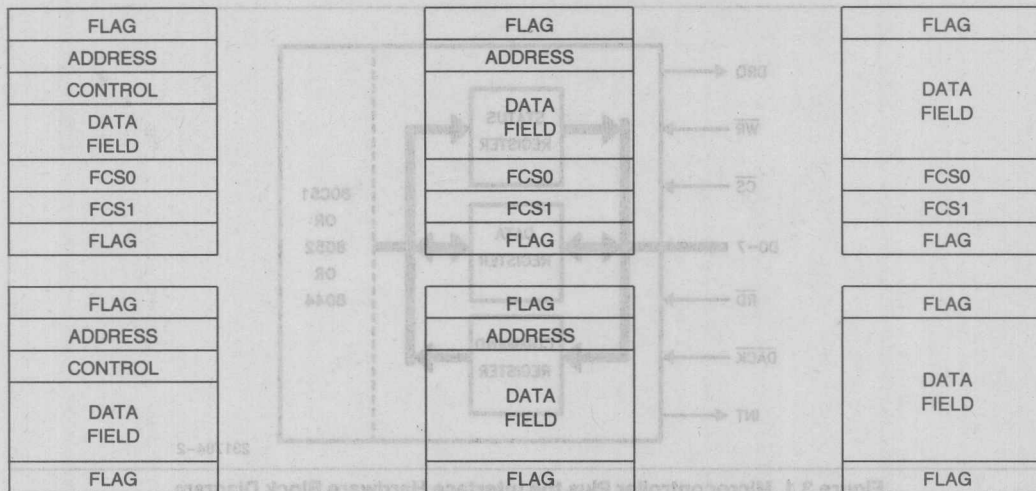


Figure 2.5. The 8044 Frame Formats

The SIU is called an intelligent channel because it responds to some SDLC commands automatically without the CPU intervention when it is set in auto mode. These automatic responses substantially reduce the communication software. Figure 2.4 gives the commands and the automatic responses.

The 8044 supports many types of frames including the standard SDLC format. Figure 2.5 shows the types of frames the 8044 can transmit and receive. If a format with an address byte is chosen, the 8044 performs address filtering during reception and transmits the contents of the station address register during transmission automatically. If a format with FCS bytes is chosen, the 8044 performs Cyclic Redundancy Check (CRC) during reception and calculates the FCS bytes during transmission of a frame in hardware. Two preamble bytes (PFS) may optionally be added to the frames. Formats that include the station address and the control byte are supported both in the auto and flexible modes.

3.0 80186/MICROCONTROLLER INTERACTION

The 80186 communicates with the microcontroller (8044, 80C51 or 8052) through the system's memory and the Command/Data and Status registers. The CPU creates a data structure in the memory, programs the DMA controller with the start address and byte count of the block, and issues a command to the microcontroller. A hypothetical block diagram of a microcontroller when used with the interface hardware is given in Figure 3.1.

Chip select and interrupt lines are used to communicate between the microcontroller and the host. The inter-

rupt is used by the microcontroller to draw the 80186's attention. The Chip Select is used by the 80186 to draw the microcontroller's attention to a new command.

There are two kinds of transfers over the bus: Command/Status and data transfers. Command/Status transfers are always performed by the CPU. Data transfers are requested by the microcontroller and are typically performed by the DMA controller.

The CPU writes commands using CS and WR signals and interrupts the microcontroller. The microcontroller reads the command, decodes it and performs the necessary actions. The CPU reads the status register using CS and RD signals (see Figure 4.1).

To initiate a command like TRANSMIT or CONFIGURE, a write operation to the microcontroller is issued by the CPU. A read operation from the CPU gives the status of the microcontroller. Section 5 discusses details on these commands and the status.

Any parameters or data associated with the command are transferred between the system memory and the microcontroller using DMA. The 80186 prepares a data block in memory. Its first byte specifies the length of the rest of the block. The rest of the block is the information field. The CPU programs the DMA controller with the start address of the block, length of the block and other control information and then issues the command to the microcontroller.

When the microcontroller requires access to the memory for parameter or data transfer, it activates the 80186 DMA request line and uses the DMA controller to achieve the data transfer. Upon completion of an operation, the microcontroller interrupts the 80186. The CPU then reads results of the operation and status of the microcontroller.

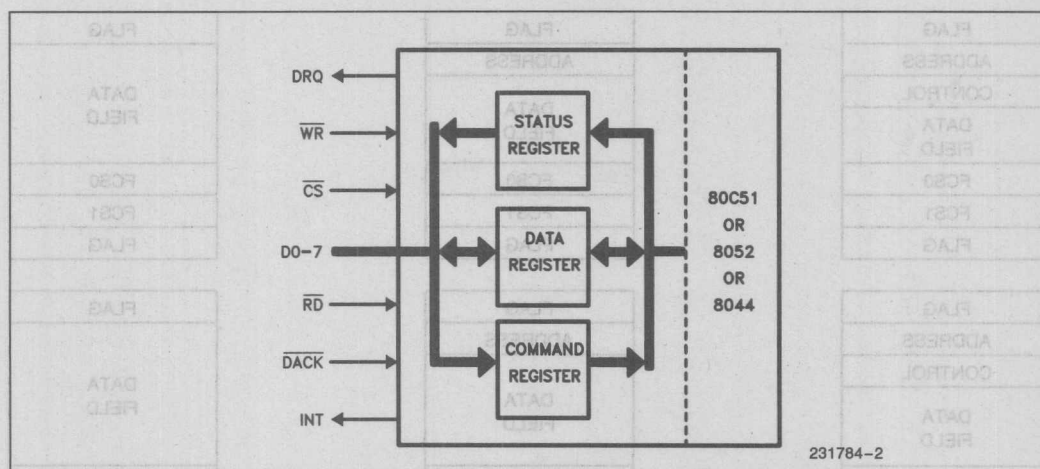


Figure 3.1. Microcontroller Plus the Interface Hardware Block Diagram

4.0 SYSTEM INTERFACE

There are two kinds of transfers over the bus: command/status and data transfers. The command/status transfers are always initiated and performed by the 80186. The data transfers are requested by the microcontroller using the DMA request (DRQ) line. In relatively slow systems the 80186 might also perform the data transfers. In that case, the request from the microcontroller will serve as an interrupt to the CPU. This mode of operation depends on the serial data rate.

The system interface performs command/status transfers, data/parameter transfers, and interrupts. This section describes the interface between the 80186 and a microcontroller shown in Figure 1.1. Section 6 describes the interface hardware.

4.1 Command/Status Transfers

The 80186 controls the microcontroller by writing into the command/data register and reading from the status register. The CPU writes a command by activating the chip select (PCSO), putting the command onto the data bus, and activating the WR signal. The command byte is latched into the command/data register, and the microcontroller is interrupted. In the interrupt service routine, the microcontroller reads the command byte from the command/data register, decodes the command byte, and activates the DRQ for data or parameter transfer if the decoded command requires such transfer.

At the end of parameter transfer the microcontroller updates the status register and interrupts the 80186.

4.2 Data/Parameter Transfer

Data/parameter transfers are controlled by a pair of REQUEST/ACKNOWLEDGE lines: DMA Request line (DRQ) and DMA Acknowledge line (DACK). Data and parameters are transferred via the Command/Data register to or from memory.

In order to request a transfer from memory, the microcontroller activates the DRQ pin. The DRQ signal goes active after a read operation by the microcontroller. In response, the 80186 DMA controller performs a byte transfer from the memory to the Command/Data register. Data is transferred on the bus and written into the Command/Data register on the rising edge of the 80186 WR signal (MWR), which is activated by the DMA controller. Figure 4.2 shows the write timing.

In order to request a transfer to memory, the microcontroller activates the DRQ signal and outputs the data into the Command/Data latch. When the microcontroller WR signal goes active, DRQ is set. In response, the DMA performs the data transfer and resets the DRQ signal. Figure 4.3 shows the read timing.

4.3 Interrupt

The microcontroller reports on completion of an event by updating the status register and raising the interrupt signal assuming this signal is initially low. The interrupt is cleared by the command from the CPU where

the INTERRUPT ACKNOWLEDGE bit is set (MD7). The INTA bit is the most significant bit of the command byte. Figure 4.4 and 4.5 show the interrupt timing. Note that it is the responsibility of the CPU to clear the interrupt in order to prevent a deadlock.

80186 Pin Name			Function
CS	RD	WR	
1	X	X	No Transfer to/from Command/Status
0	1	1	
0	0	0	Illegal
0	0	1	Read from Status Register
0	1	0	Write to Command/Data Register
DACK	RD	WR	
1	X	X	No Transfer
0	1	1	
0	0	0	Illegal
0	0	1	Data Read from DMA Channel
0	1	0	Data Write to DMA Channel

NOTE:

Only one of CS, DACK may be active at any time.

Figure 4.1. Data Bus Control Signals and Their Functions

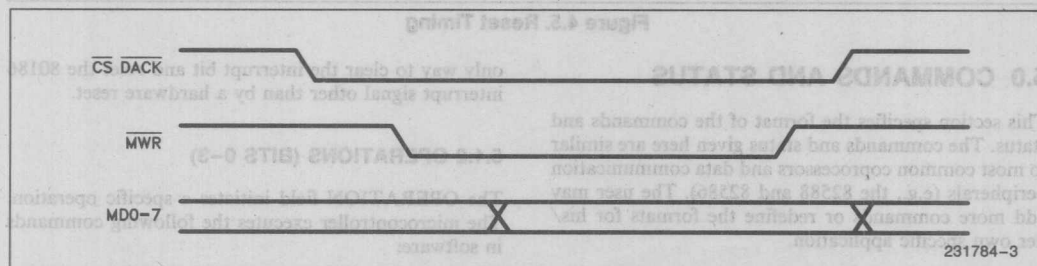


Figure 4.2. Write Timing

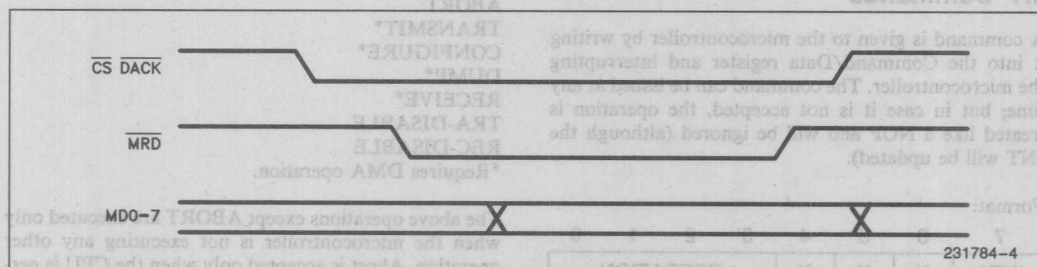


Figure 4.3. Read Timing

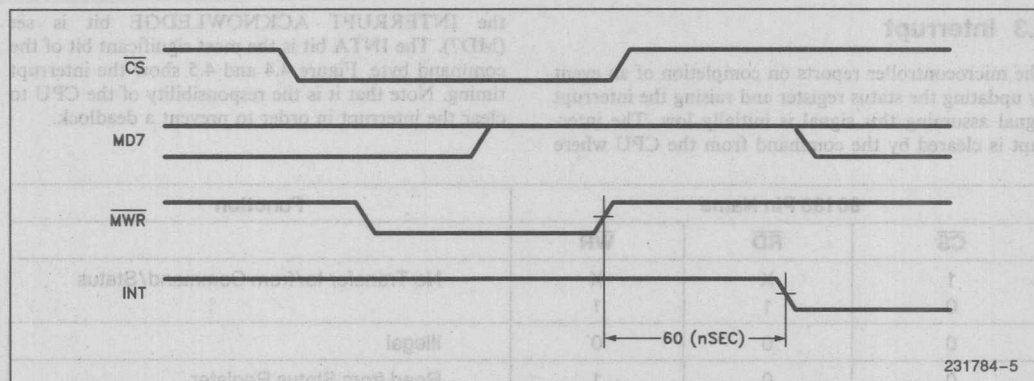


Figure 4.4. Interrupt Timing (Going Inactive)

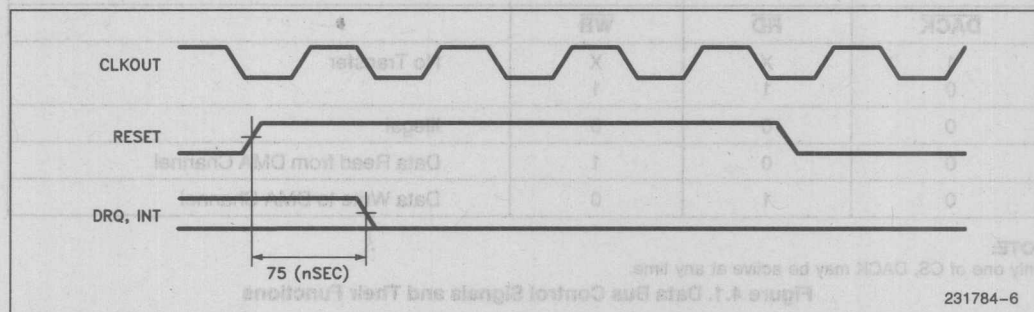


Figure 4.5. Reset Timing

5.0 COMMANDS AND STATUS

This section specifies the format of the commands and status. The commands and status given here are similar to most common coprocessors and data communication peripherals (e.g., the 82588 and 82586). The user may add more commands or redefine the formats for his/her own specific application.

5.1 Commands

A command is given to the microcontroller by writing it into the Command/Data register and interrupting the microcontroller. The command can be issued at any time; but in case it is not accepted, the operation is treated like a NOP and will be ignored (although the INT will be updated).

Format:

7	6	5	4	3	2	1	0
INTA	X	X	X				OPERATION

5.1.1 ACKNOWLEDGING INTERRUPT (BIT 7)

The INTA bit, if set, causes the interrupt hardware signal and the interrupt bit to be cleared. This is the

only way to clear the interrupt bit and reset the 80186 interrupt signal other than by a hardware reset.

5.1.2 OPERATIONS (BITS 0-3)

The OPERATION field initiates a specific operation. The microcontroller executes the following commands in software:

NOP
ABORT
TRANSMIT*
CONFIGURE*
DUMP*
RECEIVE*
TRA-DISABLE
REC-DISABLE
*Requires DMA operation.

The above operations except ABORT are executed only when the microcontroller is not executing any other operation. Abort is accepted only when the CPU is performing a DMA operation.

Operations that require parameter transfer (e.g., CONFIGURE and DUMP) or data transfer (e.g., TRANSMIT and RECEIVE) are called parametric operations. The remaining are called non-parametric operations.

An operation is initiated by writing into the command register. This causes the microcontroller to execute the command decode instructions. Some of the operations cause the microcontroller to read parameters from memory. The parameters are organized in a block that starts with an 8-bit byte count. The byte count specifies the length of the rest of the block. Before beginning the operation, the DMA pointer of the DMA channel must point to the byte count. There is no restriction on the memory structure of the parameter block as long as the microcontroller receives the next byte of the block for every DMA request it generates. Transferring the bytes is the job of the 80186 DMA controller.

The microcontroller requests the byte-count and determines the length of the parameter block. It then requests the parameters.

Upon completion of the operation, (when interrupt is low) the microcontroller updates the status, raises the interrupt signal, and goes idle.

NOP

This operation does not affect the microcontroller. It has no parameters and no results.

ABORT

This operation attempts to abort the completion of an operation under execution. It is valid for CONFIGURE, TRANSMIT, DUMP, and RECEIVE. It is ignored for any of the above if transfer of parameters has already been accomplished. The microcontroller, upon reception of the ABORT command, stops the DMA operation and issues an Execution-Aborted interrupt.

TRANSMIT

This operation transmits one message. A message may be transmitted as an SDLC frame by the 8044, or in ASYNC protocol by the 80C51 or the 8052 serial port.

Figure 5.1 shows the format of the Transmit block. A typical transmit operation parameter block includes the destination address and the control byte in the information field. As an example, see the 8044 transmit block in Figure 7.2.

7	6	5	4	3	2	1	0
BYTE COUNT							
FIRST INFO BYTE							
LAST INFO BYTE							

Figure 5.1. Format of Transmit Block

The transmit operation will either complete the execution or be aborted by a specific ABORT operation. A Transmit-Done or Execution-Aborted interrupt is issued upon completion of this operation.

CONFIGURE

This operation configures the microcontroller's internal registers. The length and the part of the configuration block that is modified are determined by the first two bytes of the command parameter (see Figure 5.2). The FIRST BYTE specifies the first register in the configure block that will be configured, and the BYTE COUNT specifies the number of registers that will be configured starting with the FIRST BYTE. For example, if the FIRST BYTE is 1 and the BYTE COUNT is the length of the configure block, then all of the registers are updated. If FIRST BYTE is 4 and BYTE COUNT is 2, then only the fourth register in the configure block is updated. Minimum byte count is 2.

7	6	5	4	3	2	1	0
BYTE COUNT							
FIRST BYTE							
FIRST REGISTER							
LAST REGISTER							

Figure 5.2. Format of Configure Block

A Configure-Done interrupt is issued when the operation is done unless ABORT was issued during the DMA operation.

DUMP

This operation causes dumping of a set of microcontroller internal registers to system memory. Figure 7.4 shows the format of the 8044 DUMP block.

The DUMP operation will either complete the execution or be aborted by a specific ABORT operation. A Dump-Done or Execution-Aborted interrupt is issued upon completion of this operation.

RECEIVE

This operation enables the reception of frames. It is ignored if the microcontroller's serial channel is already in reception mode.

The serial port receives only frames that pass the address filtering. The microcontroller transfers the received information and the byte count to the system memory using DMA. The completion of frame reception causes a Receive-Done event.

REC-DISABLE

This operation causes reception to be disabled. If transfer of data to the 80186 memory has already begun, then it is treated like the ABORT command. This operation has no parameters. REC-DISABLE is accepted only when the microcontroller's serial port is in receive mode.

TRA-DISABLE

This operation causes the transmission process to be aborted. If the microcontroller is fetching data from 80186 memory, then it is treated like the ABORT command. This operation has no parameters. It is accepted only when the serial port is in transmit mode.

5.1.3 ILLEGAL COMMANDS

Parametric and non-parametric commands except ABORT will be rejected (interrupt will not be set) if the microcontroller is already executing a command.

ABORT is rejected if issued when the microcontroller is not requesting DMA operation, or a non-Parametric execution is performed, or transfer of parameters/data has already been accomplished.

DMA operations shall not be aborted by any non-parametric or parametric command except by the ABORT command.

REC-DISABLE and TRA-DISABLE will not be accepted if the serial channel is idle.

5.2 Status

The microcontroller provides the information about the last operation that was executed, via the status register.

The microcontroller reports on these events by updating a status register and raising the INTERRUPT signal. Information from the status register is valid provided the interrupt signal is high or bit 0 of the status being read is set.

Format:

7	6	5	4	3	2	1	0
CTS*	RTS*	E	EVENT	DMA	INT		

*8044 only

5.2.1 INTERRUPT (BIT 0)

The interrupt bit is set together with the hardware interrupt signal. Setting the INT bit indicates the occurrence of an event. This bit is cleared by any command whose INTA bit is set. Status is valid only when this bit is set.

5.2.2 DMA OPERATION (BIT 1)

The DMA bit, when set, indicates that a DMA operation is in progress. This bit is set if the command received by the microcontroller requires data or parameter transfer. If this bit is clear, DRQ will be inactive. The DMA bit, when cleared, indicates the completion of a DMA operation.

5.2.3 ERROR (BIT 5)

The E bit, if set, indicates that the event generated for the operation that was completed contains a warning, or the operation was not accepted.

5.2.4 REQUEST TO SEND (BIT 6)

The RTS bit, if clear, indicates that the serial channel is requesting a transmission.

5.2.5 CLEAR TO SEND (BIT 7)

The CTS bit indicates that, if the RTS bit is clear, the serial port is active and transmitting a frame.

5.2.6 EVENT (BITS 2-4)

The event field specifies why the microcontroller needs the attention of the 80186.

The following events may occur:

CONFIGURE-DONE
TRANSMIT-DONE
DUMP-DONE
RECEIVE-DONE
RECEPTION-DISABLED
TRANSMISSION-DISABLED
EXECUTION-ABORTED

CONFIGURE-DONE

This event indicates the completion of a CONFIGURE operation.

TRANSMIT-DONE

This event indicates the completion of the TRANSMIT operation.

If the E bit is set, it indicates that the transmit buffer was already full.

DUMP-DONE

This event indicates that the DUMP operation is completed.

RECEIVE-DONE

This event indicates that a frame has been received and stored in memory.

The format of the received message is indicated in Figure 5.3.

7	6	5	4	3	2	1	0
FIRST INFO BYTE							
LAST INFO BYTE							
RECEIVED BYTE COUNT							

Figure 5.3. Format of Receive Block

Following the byte count, a few more bytes relating to the received frame such as the source address and the control byte may be transferred to the system memory using DMA. As an example, see the 8044 receive block in Figure 7.3.

Note that the format of a frame received by the microcontroller serial channel is configured by the CONFIGURE command.

If the E bit is set, buffer overrun has occurred.

RECEPTION-DISABLED

This event is issued as a result of a RCV-DISABLE operation that causes part of a frame to be disabled.

If the E bit is set, the serial port was already disabled, and the RCV-DISABLE is not accepted.

TRANSMISSION-DISABLED

This event is issued as a result of a TRA-DISABLE operation that causes transmission of a frame to be disabled.

The E bit, if set, indicates that the TRA-DISABLE operation was not accepted since the serial port was already idle, or transmission of a frame has already been accomplished.

EXECUTION-ABORTED

This event indicates that the execution of the last operation was aborted by the ABORT command.

If the E bit is set, ABORT was issued when the microcontroller was not executing any commands.

6.0 HARDWARE DESCRIPTION

The interface hardware shown in Figures 6.1 and 6.2 are identical. The difference is the status register. In Figure 6.2, an external latch is used to latch the status byte. This hardware is recommended if an extra I/O port on the microcontroller is required for some other applications, or external program and data memory is required for the microcontroller. The hardware shown in Figure 6.1 makes use of one of the microcontroller's I/O ports (Port 1) to latch the status to minimize hardware. The discussion of Sections 1 through 5 apply to both schematics.

6.1 Reset

After an 80186 hardware reset, the microcontroller is also reset. The on-chip registers are initialized as explained in the Intel Microcontroller Handbook. The reset signal also clears the 80186 interrupt and the microcontroller interrupt signals by resetting FF3 (Flip-Flop 3) and FF2 (Flip-Flop 2). Figure 4.5 shows the RESET timing.

6.2 Sending Commands

A bidirectional latched transceiver (74ALS646) is used for the Command/Data register. When the 80186 writes a command to the Command/Data register, it interrupts the microcontroller. The interrupt is generated only when bit 7 (INTA) of the command byte is set. When the 80186 PCS0 and WR signals go active to write the command, FF2 will be set and FF3 will be cleared. The output of FF3 is the interrupt to the 80186 and the INT status bit. The INT bit is cleared immediately to indicate that the status is no longer valid. The output of FF2 is the interrupt to the microcontroller. A high to low transition on this line will interrupt the microcontroller. The interrupt signal will be cleared as soon as the microcontroller reads the command from the Command/Data register.

6.3 DMA Transfers

In the interrupt service routine the command is decoded. If it requires a DMA transfer, the microcontroller sets the DMA bit of the status register which activates the DMA request signal. DRQ active causes the 80186 on-chip DMA to perform a fetch and a deposit bus cycle. The first DMA cycle clears the DRQ signal (FF1 is cleared). When the microcontroller performs a read or write operation, the output of the FF1 will be set, and DRO goes active again.

The DMA controller transfers a byte from system memory to the Command/Data register. Data is latched when the 80186 PCS1 and WR signals go active. PCS1 and WR active also clear FF1. The microcontroller monitors the output of FF1 by polling the P3.3 pin. When FF1 is cleared the microcontroller reads the byte from the Command/Data register. The P3.3 pin is also the interrupt pin. If a slow rate of transfer is acceptable, every DMA transfer can be interrupt driven to allow the microcontroller to perform other tasks.

The DMA controller transfers a byte from the Command/Data register to system memory by activating

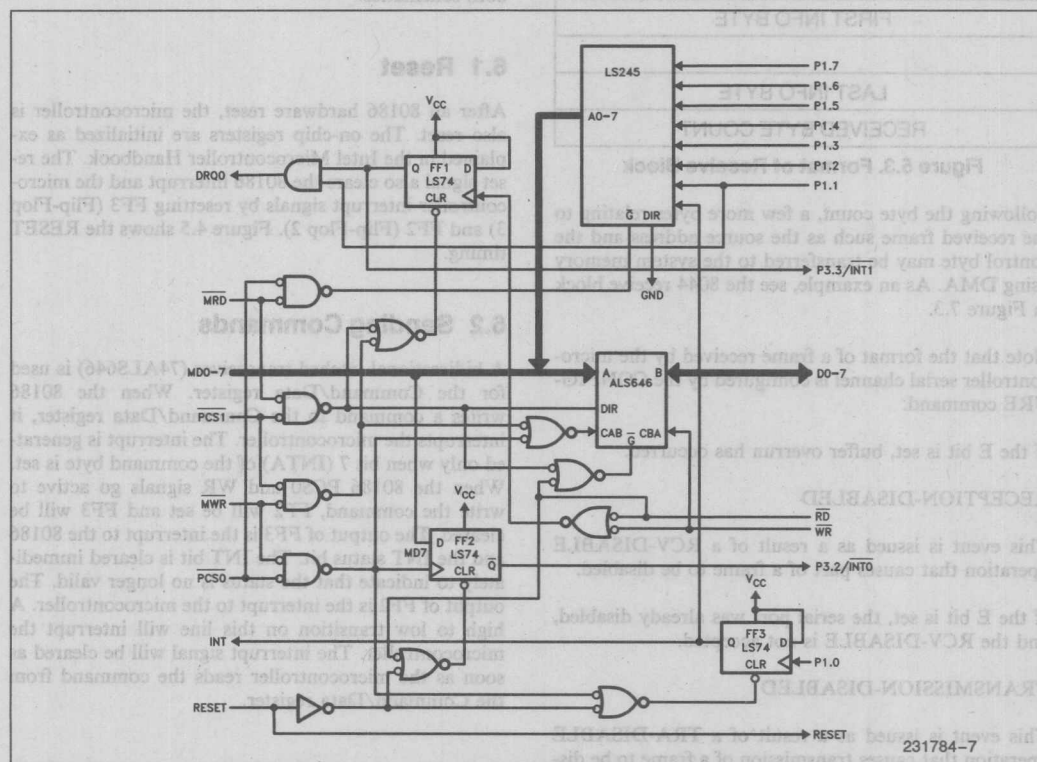
the 80186 PCS1 and RD signals. PCS1 and RD active also clear FF1. When FF1 is cleared the microcontroller writes the next byte to the Command/Data register.

When all the data is transferred, the microcontroller clears the DMA status bit to disable DRQ. It then updates the status, sets the INT bit, and interrupts the 80186.

If the interface hardware in Figure 6.1 is used P1.1 is the DMA status bit and P1.0 is the INT bit. The microcontroller enables or disables them by writing to port 1. In Figure 6.2, DRQ or INT is disabled or enabled by writing to the 74LS374 status register. Note that the INT status bit is cleared by the hardware when the 80186 writes a command.

6.4 Reading Status

The command is written and the status is read with the same chip select (PCS0), although the status is read through the 74LS245 transceiver and the command is written to the Command/Data register.



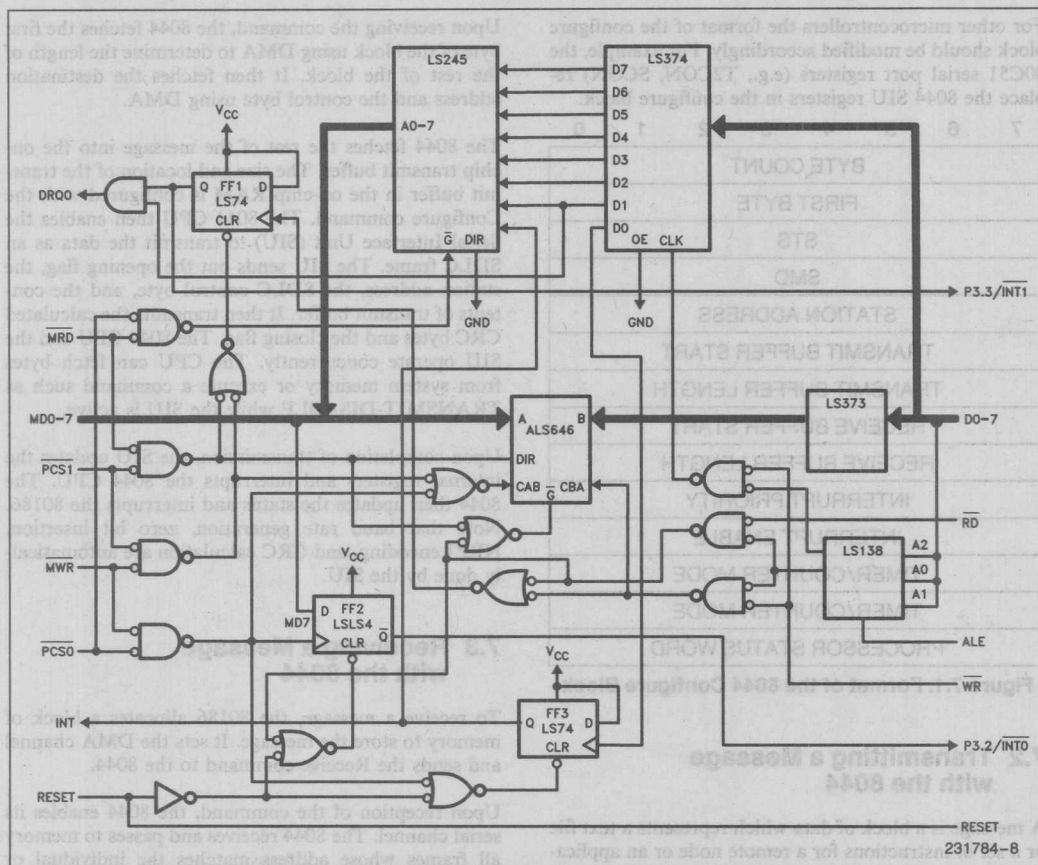


Figure 6.2. Hardware Interface

The microcontroller updates the status byte whenever a change occurs in the status and outputs the result to the status register. In order to read status, the 80186 activates the PCS0 line, and then activates the RD line. The contents of the status are put on the data bus, through the 74LS245 transceiver.

For systems that require two DMA channels, a second pair of DRQ1/DACK1 signals may easily be added to the hardware. In that case one of the status bits (DMA2) ANDed with the output of FF1 will serve as the second DMA request signal (DRQ1). DACK1 can be generated with the 80186 PCS2.

7.0 8044/80186 INTERFACE

This section shows how to make use of the status and commands described in section 5 and the hardware given in Figure 6.1 to interface the 80186 with the 8044. The 8044 code to implement these functions is shown in Appendix A.

7.1 Configuring the 8044

This operation configures the 8044 registers. The format of the configure block is shown in Figure 7.1. The part of the configuration block that is modified is determined by the first two bytes of the command parameter. The FIRST BYTE specifies the first register in the configure block that will be configured, and the BYTE COUNT specifies the number of registers that will be configured starting with the FIRST BYTE. For example, if the FIRST BYTE is 1 and the BYTE COUNT is 13, then all of the registers are updated. If FIRST BYTE is 4 and BYTE COUNT is 2, then transmit buffer start register is configured.

The configure command performs the following: 1) configures the interrupts and assigns their priorities; 2) assigns the start address and length of the transmit and receive buffers; 3) sets the station address; 4) sets the clock option and the frame format.

For other microcontrollers the format of the configure block should be modified accordingly. For example, the 80C51 serial port registers (e.g., T2CON, SCON) replace the 8044 SIU registers in the configure block.

7	6	5	4	3	2	1	0
BYTE COUNT							
FIRST BYTE							
STS							
SMD							
STATION ADDRESS							
TRANSMIT BUFFER START							
TRANSMIT BUFFER LENGTH							
RECEIVE BUFFER START							
RECEIVE BUFFER LENGTH							
INTERRUPT PRIORITY							
INTERRUPT ENABLE							
TIMER/COUNTER MODE							
TIMER/COUNTER MODE							
PROCESSOR STATUS WORD							

Figure 7.1. Format of the 8044 Configure Block

7.2 Transmitting a Message with the 8044

A message is a block of data which represents a text file or a set of instructions for a remote node or an application program which resides on the 8044 program memory. A message can be a frame (packet) by itself or can be comprised of multiple frames. An SDLC frame is the smallest block of data that the 8044 transmits. The 8044 can receive commands from the 80186 to transmit and receive messages. The 8044 on-chip CPU can be programmed to divide messages into frames if necessary. Maximum frame size is limited by the transmit or receive buffer.

To transmit a message, the 80186 prepares a transmit data block in memory as shown in Figure 7.2. Its first byte specifies the length of the rest of the block. The next two bytes specify the destination address of the node the message is being sent to and the control byte of the message. The 80186 programs the DMA controller with the start address of the block, length of the block and other control information and then issues the Transmit command to the 8044.

Upon receiving the command, the 8044 fetches the first byte of the block using DMA to determine the length of the rest of the block. It then fetches the destination address and the control byte using DMA.

The 8044 fetches the rest of the message into the on-chip transmit buffer. The size and location of the transmit buffer in the on-chip RAM is configured with the Configure command. The 8044 CPU then enables the Serial Interface Unit (SIU) to transmit the data as an SDLC frame. The SIU sends out the opening flag, the station address, the SDLC control byte, and the contents of transmit buffer. It then transmits the calculated CRC bytes and the closing flag. The 8044 CPU and the SIU operate concurrently. The CPU can fetch bytes from system memory or execute a command such as TRANSMIT-DISABLE while the SIU is active.

Upon completion of transmission, the SIU updates the internal registers and interrupts the 8044 CPU. The 8044 then updates the status and interrupts the 80186. Note that baud rate generation, zero bit insertion, NRZI encoding, and CRC calculation are automatically done by the SIU.

7.3 Receiving a Message with the 8044

To receive a message, the 80186 allocates a block of memory to store the message. It sets the DMA channel and sends the Receive command to the 8044.

Upon reception of the command, the 8044 enables its serial channel. The 8044 receives and passes to memory all frames whose address matches the individual or broadcast address and passes the CRC test.

The SIU performs NRZI decoding and zero bit deletion, then stores the information field of the received frame in the on-chip receive buffer. At the end of reception, the CPU requests the transfer of data bytes to 80186 memory using DMA. After transferring all the bytes, the 8044 transfers the data length, source address, and control byte of the received frame to the memory (see Figure 7.3). Upon completion of the transfers, the 8044 updates the status register and raises the interrupt signal to inform the 80186.

If the SIU is not ready when the first byte of the frame arrives, then the whole frame is ignored. Disabling reception after the first byte was passed to memory causes the rest of the frame to be ignored and an interrupt with Receive-Aborted event to be issued.

Full (TBF) bit. During transmission, if the TBF bit is cleared, the SIU will discontinue the transmission and interrupt the 8044 CPU.

The RECEIVE-DISABLE command causes the 8044 to clear the Receive Buffer Empty (RBE) bit. The SIU aborts the reception, if the RBE bit is cleared by the CPU.

When transmission or reception of a frame is discontinued, the SIU interrupts the 8044 CPU. The CPU then updates the status and interrupts the 80186.

7.7 Handling Interrupts

When the 80186 sends a command, it sets the 8044 external interrupt flag. The 8044 services the interrupt at its own convenience. In the interrupt service routine the 8044 executes the appropriate instructions for a given command. During execution of a command the 8044 ignores any command, except ABORT, sent by the 80186 (see section 5.1.2). This is accomplished by clearing the interrupt flag before the 8044 returns from the interrupt service routine. During DMA operations the 8044 sets the external interrupt to high priority. An interrupt with high priority can suspend execution of an interrupt service routine with low priority. The ABORT command given by the 80186 will interrupt the execution of the DMA transfer in progress. Upon completion of ABORT, execution of the last operation will not be resumed (see Appendix A). Note that any other command given during the DMA operation will also abort the operation in progress and should be avoided.

8.0 8044 IN EXPANDED OPERATION

To increase the number of information bytes in a frame, the 8044 can be operated in Expanded mode. In Expanded operation the system memory can be used as the transmit and receive buffer instead of the 8044 internal RAM. AP-283, "Flexibility in Frame Size Operation with the 8044", describes Expanded operation in detail.

8.1 Transmitting a Message in Expanded Operation

In Expanded operation the 8044 transmits the frame while it is fetching the data from the system memory using DMA. An internal transmit buffer is not necessary. The system memory can be used as the transmit buffer by the 8044.

Upon receiving the Transmit command, the 8044 enables the SIU and fetches the first data byte from the Command/Data register. The SIU transmits the opening flag, station address, and the control byte if the frame format includes these fields. It then transmits the

fetches data. The 8044 CPU fetches the next byte while the previously fetched byte is being transmitted by the SIU. The CPU fetches the remaining bytes using DMA, then the SIU transmits them simultaneously until the end of message is reached. The SIU then transmits the FCS bytes, the closing flag and interrupts the 8044 CPU. The 8044 updates the status with the Transmit-Done event and interrupts the 80186. If the DMA does not keep up with transmission, the transmission is an underrun.

8.2 Receiving a Message in Expanded Operation

In Expanded operation the DMA controller transfers data to the system memory while the 8044 SIU is receiving them.

To receive a message, the 80186 allocates a block of memory for storing the message. It sets the DMA channel and sends the Receive command to the 8044.

Upon reception of the command, the 8044 enables its serial channel and waits for a frame. The SIU performs flag detection, address filtering, zero bit deletion, NRZI decoding, and CRC checking as it does in Normal operation.

After the SIU receives the first byte of the frame, the 8044 CPU requests the transfer of the byte to memory using DMA. The 80186 DMA moves the information byte into the system memory while the SIU is receiving the next byte. The next byte is transferred to the memory after the SIU receives it. When the entire frame is received, the SIU checks the received Frame Check Sequence bytes. If there is no CRC error, the SIU updates the 8044 registers and interrupts the 8044 CPU. The CPU updates the status and interrupts the 80186.

9.0 CONCLUSION

This application note describes an efficient way to interface the 80186 and the 80188 microprocessors to the Intel 8-bit microcontrollers like the 80C51, 8052, and 8044. To illustrate this point the 80186 microprocessor interface to the 8044 microcontroller based serial communication chip was described. The hardware interface given here is very general and can interface the 8-bit microcontrollers to a variety of Intel microprocessors and DMA controllers. The microcontrollers with this interface hardware have the same benefits as both the Intel UPI-41/42 family and data communication peripheral chips such as the 82588 and the 82568 LAN controllers. Like the Intel UPI chips, they can be easily interfaced to microprocessors, and like the data communication peripherals, they execute high level commands. A similar approach can be used to interface Intel microprocessors to the 16-bit 8096 microcontroller.

APPENDIX A SOFTWARE

The software modules shown here implement the execution of commands and status explained in sections 5 and 7. The 80186 software provides procedures to send commands and read status. The 8044 software decodes and executes the commands, updates the status, and interrupts the 80186. The procedures given here are called by higher level software drivers. For example, an 80186 application program may use the Transmit command to send a block of data to an application program that resides in the 8044 ROM or in another remote node. The application programs and the drivers that perform the communication tasks run asynchronously since all communication tasks are interrupt-driven.

Figure A-1 shows how to assign the ports and control registers for an 80186-based system. The software is written for an Intel iSBC® 186/51 computer board. The 8044 hardware is connected to the computer board iSBX™ connector.

Figure A-2 shows the 80186 command procedures. These procedures are used by the data link driver.

Figure A-3 shows how the DMA controller is loaded and initialized for data and parameter transfer from the 80186 memory to the 8044. This procedure is used by the TRANSMIT and CONFIGURE commands.

Figure A-4 shows how the DMA controller is loaded and initialized for data and parameter transfer from the 8044 to the 80186 memory. This procedure is used by the RECEIVE and DUMP commands.

Figure A-5 shows an interrupt service routine which handles interrupts resulting from various events. Note that this routine is not complete. The user should write the software to respond to events.

Figure A-6 shows an example of the 80186 software. It shows how to start various operations. This is not a data link driver, but it gives the procedures needed to write a complete driver.

Figure A-7 shows how to initialize the 8044. The user application program should be inserted here.

Figures A-8 through A-13 show the 8044 external interrupt service routine. In this routine a command received from the 80186 is decoded, and one of the command procedures shown in Figures A-9 through A-13 is executed.

Figure A-14 shows the serial channel (SIU) interrupt service routine. Note that execution of TRANSMIT, RECEIVE, and TRANSMIT-DISABLE commands are completed in this routine.

NAME COM_DRIVER

```

; ** 80186 SOFTWARE FOR THE 80186/MICROCONTROLLER INTERFACE
; * 8044 BOARD CONNECTED TO THE SBX1 OF THE SEC 186/51 BOARD.
; * SBX1 INTO TIED TO 80130 IR[0-7]. CONNECT JUMPER 30 TO 46.
; * 80186 DMA CHANNEL 1 USED. CONNECT JUMPER 202 TO 203.

```

```

TRUE      EQU  OFFFFH
FALSE     EQU  0H

```

; 8044 REGISTERS

```

CMD_44    EQU  080H    ; ADDRESS OF THE COMMAND REGISTER
ST_44     EQU  080H    ; ADDRESS OF THE STATUS REGISTER
DATA_44    EQU  0D4H    ; ADDRESS OF THE DATA REGISTER

```

; EVENTS

```

CON_DONE   EQU  01H    ; CONFIGURE DONE
TRA_DONE   EQU  02H    ; TRANSMIT DONE
DUM_DONE   EQU  03H    ; DUMP DONE
REC_DONE   EQU  04H    ; RECEIVE DONE
REC_DISA   EQU  05H    ; RECEPTION DISABLE
TRA_DISA   EQU  06H    ; TRANSMISSION DISABLE
ABO_DONE   EQU  07H    ; EXECUTION_ABORTED

```

; COMMANDS (INTA=1)

```

ABO_CMD    EQU  080H    ; ABORT
REC_DIS_CMD EQU  081H    ; RECEIVE DISABLE
XMIT_DIS_CMD EQU  082H    ; TRANSMIT DISABLE
REC_CMD    EQU  083H    ; RECEIVE
TRA_CMD    EQU  084H    ; TRANSMIT
DUM_CMD    EQU  085H    ; DUMP
CON_CMD    EQU  086H    ; CONFIGURE
NOP_CMD    EQU  087H    ; NOP

```

; 80186 DMA CHANNEL 1 REGISTERS

```

SL_DMA1    EQU  OFFD0H  ; SOURCE ADDRESS (LO WORD)
SH_DMA1    EQU  OFFD2H  ; SOURCE ADDRESS (HI WORD)
DL_DMA1    EQU  OFFD4H  ; DESTINATION ADDRESS (LO WORD)
DH_DMA1    EQU  OFFD6H  ; DESTINATION ADDRESS (HI WORD)
CNT_DMA1   EQU  OFFD8H  ; TRANSFER COUNT ADDRESS
CTL_DMA1   EQU  OFFDAH  ; CONTROL ADDRESS

```

; 80186 INTERRUPT CONTROLLER REGISTERS

```

CTLO_INTR  EQU  OFF38H  ; INT 0 CONTROL ADDRESS
CTLI_INTR  EQU  OFF3AH  ; INT 1 CONTROL REGISTER
MASK_INTR  EQU  OFF28H  ; INT MASK REGISTER
EOI_INTR   EQU  OFF22H  ; INT EOI REGISTER
NSPEC_BIT  EQU  08000H  ; NON-SPECIFIC EOI

```

; 80130 INTERRUPT CONTROLLER REGISTERS

```

EOI_SINTR  EQU  0E0H    ; INT EOI REGISTER
MASK_SINTR EQU  0E2H    ; MASK REGISTER
RD_IRR     EQU  010H    ; COMMAND TO 80130 TO READ IRR REG
RD_ISR     EQU  011H    ; COMMAND TO 80130 TO READ ISR REG

```

```

IV_BASE    EQU  20H      ; BASE OF 80130 INT CONTROLLER VECTOR

```

231784-11

Figure A-1. Port and Register Definitions for 80186 System

```

;*****
; INTERRUPT TABLE

INTERRUPTS      SEGMENT AT 0

                ORG (IV_BASE+1)*4H

IV_INTRO LABEL   DWORD 00000000H ; IR1 VECTOR

INTERRUPTS      ENDS

;*****

STACK           SEGMENT STACK 'STACK'

THE_STACK DW    200H DUP(?)
TOS LABEL

STACK           ENDS

;*****

DATA            SEGMENT PUBLIC 'DATA'

REC_BUFFER      DB    1024 DUP(?)

CON_BUFFER      DB    08H,01H,00H,0D0H,55H,20H,05H,30H,05H

DUM_BUFFER      DB    0FH DUP(?)

TRA_BUFFER      DB    07H,55H,11H,01H,02H,03H,04H,05H

CMDN_FLAG       DW    FALSE

DATA            ENDS

```

231784-12

Figure A-1. Port and Register Definitions for 80186 System (Continued)

```

;*****
CODE            SEGMENT PUBLIC 'CODE'

ASSUME CS:CODE,
        DS:DATA,
        ES:NOTHING,
        SS:STACK

;*****

RCV_COMMAND     PROC FAR

    PUSH BP
    MOV BP,SP
    LES SI,DWORD PTR [BP+6] ; LOAD BUFFER POINTER
    MOV AX,WORD PTR [BP+10] ; LOAD BUFFER SIZE
    MOV AH,0H
    CALL REC_DMA ; CALL REC-DMA
    MOV AL,RCV_CMD ; LOAD RECEIVE COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    POP BP
    RET

RCV_COMMAND     ENDP

;*****

XMIT_COMMAND     PROC FAR

    PUSH BP
    MOV BP,SP
    LES SI,DWORD PTR [BP+6] ; LOAD BUFFER POINTER
    MOV AX,WORD PTR [BP+10] ; LOAD BUFFER SIZE
    MOV AH,0H
    CALL TRA_DMA ; CALL TRA-DMA
    MOV AL,TRA_CMD ; LOAD TRANSMIT COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    POP BP
    RET

XMIT_COMMAND     ENDP

```

231784-13

Figure A-2. Setup and Execution of Commands

```

;*****
CONF_COMMAND PROC FAR
    PUSH BP
    MOV BP,SP
    LES SI,DWORD PTR[BP+6] ; LOAD BUFFER POINTER
    MOV AX,WORD PTR[BP+10] ; LOAD BUFFER SIZE
    MOV AH,0H
    CALL TRA_DMA ; CALL TRA-DMA
    MOV AL,CON_CMD ; LOAD CONFIGURE COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    POP BP
    RET

CONF_COMMAND ENDP

;*****
DUMP_COMMAND PROC FAR
    PUSH BP
    MOV BP,SP
    LES SI,DWORD PTR[BP+6] ; LOAD BUFFER POINTER
    MOV AX,WORD PTR[BP+10] ; LOAD BUFFER SIZE
    MOV AH,0H
    CALL REC_DMA ; CALL REC-DMA
    MOV AL,DUM_CMD ; LOAD DUMP COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    POP BP
    RET

DUMP_COMMAND ENDP

;*****
XMIT_DIS_COMMAND PROC FAR
    MOV AL,XMIT_DIS_CMD ; LOAD XMIT-DIS COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    RET

XMIT_DIS_COMMAND ENDP

;*****
REC_DIS_COMMAND PROC FAR
    MOV AL,REC_DIS_CMD ; LOAD REC-DIS COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    RET

REC_DIS_COMMAND ENDP

;*****
ABOR_COMMAND PROC FAR
    MOV AL,ABO_CMD ; LOAD ABORT COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    RET

ABOR_COMMAND ENDP

;*****
NOP_COMMAND PROC FAR
    MOV AL,NOP_CMD ; LOAD NOP COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    RET

NOP_COMMAND ENDP

```

231784-15

Figure A-2. Setup and Execution of Commands (Continued)


```

;*****
; ** RECEIVE DMA
; ARGS      AX      BUFFER SIZE
;           ES:SI    BUFFER POINTER
;
REC_DMA     PROC     NEAR
    MOV     DX,CNT_DMA1      ; LOAD ADD OF TRANSFER COUNT REG
    OUT     DX,AX            ; PROGRAM TRANSFER COUNT REGISTER

    XOR     BX,BX            ; CLEAR BX
    MOV     AX,ES            ; LOAD SEG ADDRESS OF BUFFER
    SHL     AX,1             ; CALCULATE LINEAR ADDRESS OF THE BUFFER
    RCL     BX,1
    SHL     AX,1
    RCL     BX,1
    SHL     AX,1
    RCL     BX,1
    SHL     AX,1
    RCL     BX,1
    ADD     AX,SI            ; ADD THE OFFSET TO BASE
    ADC     BX,0
    MOV     DX,DL_DMA1      ; LOAD ADDRESS OF DEST POINTER (LO WORD)
    OUT     DX,AX            ; PROGRAM DEST POINTER REGISTER (LO WORD)
    MOV     AX,BX
    MOV     DX,DH_DMA1      ; LOAD ADDRESS OF DEST POINTER (HI WORD)
    OUT     DX,AX            ; PROGRAM DEST POINTER REGISTER (HI WORD)

    MOV     AX,DATA_44      ; LOAD ADDRESS OF DATA REGISTER
    MOV     DX,SL_DMA1      ; LOAD ADDRESS OF SOURCE POINTER
    OUT     DX,AX            ; PROGRAM SOURCE POINTER REGISTER (LO WORD)

    XOR     AX,AX            ; CLEAR AX
    MOV     DX,SH_DMA1      ; LOAD ADDRESS OF SOURCE POINTER (HI WORD)
    OUT     DX,AX            ; PROGRAM SOURCE POINTER REGISTER (HI WORD)

    MOV     DX,CTL_DMA1      ; LOAD ADDRESS OF CONTROL REGISTER
    MOV     AX,1010001010100110B ; LOAD THE CONTROL WORD
    OUT     DX,AX            ; PROGRAM THE CONTRL REGISTER
    RET
REC_DMA     ENDP

```

231784-16

Figure A-3. Loading and Starting the 80186 DMA Controller

```

;*****
; ** TRANSMIT DMA
; ARGS      AX      BUFFER SIZE
;           ES:SI    BUFFER POINTER
;
TRA_DMA     PROC     NEAR
    INC     AX
    MOV     DX,CNT_DMA1      ; LOAD ADD OF TRANSFER COUNT REG
    OUT     DX,AX            ; PROGRAM TRANSFER COUNT REGISTER

    XOR     BX,BX            ; CLEAR BX
    MOV     AX,ES            ; LOAD SEG ADDRESS OF BUFFER
    SHL     AX,1             ; CALCULATE LINEAR ADDRESS OF THE BUFFER
    RCL     BX,1
    SHL     AX,1
    RCL     BX,1
    SHL     AX,1
    RCL     BX,1
    SHL     AX,1
    RCL     BX,1
    ADD     AX,SI            ; ADD THE OFFSET TO BASE
    ADC     BX,0
    MOV     DX,SL_DMA1      ; LOAD ADDRESS OF SOURCE POINTER (LO WORD)
    OUT     DX,AX            ; PROGRAM SOURCE POINTER REGISTER (LO WORD)
    MOV     AX,BX
    MOV     DX,SH_DMA1      ; LOAD ADDRESS OF SOURCE POINTER (HI WORD)
    OUT     DX,AX            ; PROGRAM SOURCE POINTER REGISTER (HI WORD)

    MOV     AX,DATA_44      ; LOAD ADDRESS OF DATA REGISTER
    MOV     DX,DL_DMA1      ; LOAD ADDRESS OF DEST POINTER
    OUT     DX,AX            ; PROGRAM DEST POINTER REGISTER (LO WORD)

    XOR     AX,AX            ; CLEAR AX
    MOV     DX,DH_DMA1      ; LOAD ADDRESS OF DEST POINTER (HI WORD)
    OUT     DX,AX            ; PROGRAM DEST POINTER REGISTER (HI WORD)

    MOV     DX,CTL_DMA1      ; LOAD ADDRESS OF CONTROL REGISTER
    MOV     AX,0001011010100110B ; LOAD THE CONTROL WORD
    OUT     DX,AX            ; PROGRAM THE CONTRL REGISTER
    RET
TRA_DMA     ENDP

```

231784-17

Figure A-4. Loading and Starting the 80186 DMA Controller

```

;*****
; 80186 INTERRUPT ROUTINE
INT_186:
    PUSH AX
    PUSH DX
    MOV AX, NSPEC_BIT ; SEND NSPEC END OF INT
    MOV DX, EOI_INTR
    OUT DX, AX

    MOV AL, 01100001B
    OUT EOI_SINTR, AL

    IN AL, ST_44 ; READ THE STATUS
    AND AX, OFFH

; DECODE STATUS AND TAKE APPROPRIATE ACTION

    MOV DX, CTL_DMA1 ; DISABLE DMA
    IN AX, DX
    OR AX, 0100B
    AND AX, NOT 010B
    OUT DX, AX

    MOV CHND_FLAG, TRUE

    POP DX
    POP AX
    IRET

```

231784-18

Figure A-5. Interrupt Service Routine

```

;*****
BEGIN:
    CLI
    CLD

; SET ALL REGISTERS SMALL MODEL

    MOV SP, DATA
    MOV DS, SP
    MOV ES, SP
    MOV SP, STACK
    MOV SS, SP
    MOV SP, OFFSET TOS

; SETUP INTERRUPT VECTORS

    PUSH ES
    XOR AX, AX
    MOV ES, AX
    MOV WORD PTR ES:IV_INTRO + 0, OFFSET INT_186
    MOV WORD PTR ES:IV_INTRO + 2, CS
    POP ES

SETUP 80130 INTERRUPT CONTROLLER

    MOV AL, 00010011B ; ICW1
    OUT EOI_SINTR, AL
    MUL AL

    MOV AL, IV_BASE ; ICW2
    OUT MASK_SINTR, AL
    MUL AL

    MOV AL, 00000000B ; ICW4
    OUT MASK_SINTR, AL
    MUL AL

    MOV AL, 0FCH ; MASK
    OUT MASK_SINTR, AL

```

231784-19

Figure A-6. Example of Executing Commands

```

; SETUP 80186 INTERRUPT CONTROLLER
MOV     AX,0000000000100000B
MOV     DX,CTLO_INTR
OUT     DX,AX

MOV     DX,CTL1_INTR
IN      AX,DX
OR      AX,0000000000101000B
OUT     DX,AX

MOV     AX,000EDH
MOV     DX,MASK_INTR
OUT     DX,AX
STI     ;ENABLE INTERRUPTS

;*** SEND CONFUIRE COMMAND
PUSH    WORD PTR CON_BUFFER
PUSH    DS
PUSH    OFFSET CON_BUFFER
CALL    CONF_COMMAND
ADD     SP,3*2

; WAIT FOR END OF COMMAND

WAIT1:  CMP     CMND_FLAG,TRUE
JNE     WAIT1
MOV     CMND_FLAG,FALSE

;*** SEND DUMP COMMAND
PUSH    WORD PTR DUM_BUFFER
PUSH    DS
PUSH    OFFSET DUM_BUFFER
CALL    DUMP_COMMAND
ADD     SP,3*2

WAIT2:  CMP     CMND_FLAG,TRUE
JNE     WAIT2
MOV     CMND_FLAG,FALSE

;*** SEND TRANSMIT COMMAND
PUSH    WORD PTR TRA_BUFFER
PUSH    DS
PUSH    OFFSET TRA_BUFFER
CALL    XMIT_COMMAND
ADD     SP,3*2

WAIT3:  CMP     CMND_FLAG,TRUE
JNE     WAIT3
MOV     CMND_FLAG,FALSE

;*** SEND RECEIVE COMMAND
PUSH    WORD PTR REC_BUFFER
PUSH    DS
PUSH    OFFSET REC_BUFFER
CALL    REC_V_COMMAND
ADD     SP,3*2

WAIT4:  CMP     CMND_FLAG,TRUE
JNE     WAIT4
MOV     CMND_FLAG,FALSE

CODE    ENDS
END     BEGIN

```

231784-20

231784-21

Figure A-6. Example of Executing Commands (Continued)

```

$DEBUG NOMOD51
$INCLUDE (REG44.PDF)

; THE 8044 SOFTWARE DRIVER FOR THE 80186/8044 INTERFACE.

        ORG 00H          ; LOCATIONS 00 THRU 26H ARE USED
        SJMP INIT        ; BY INTERRUPT SERVICE ROUTINES.
        ORG 03H          ; VECTOR ADDRESS FOR EXT INTO.
        JMP EINT0
        ORG 23H          ; VECTOR ADDRESS FOR SERIAL INT
        JMP SIINT

;***** INITIALIZATION *****

        ORG 26H
INIT:    MOV TCON,#00000001B ; EXT INTO: EDGE TRIGGER
        MOV IE,#00010001B  ; SI-EX0=1
        CLR P1.1           ; CLEAR DRQ STATUS BIT
        SETB EA            ; ENABLE INTERRUPTS
DOT:     SJMP DOT          ; WAIT FOR AN INTERRUPT

```

231784-22

Figure A-7. Initialization Routine

```

;*****EXTERNAL INTERRUPT 0 *****
EINT0:   CLR P1.5          ; CLEAR THE E BIT
        MOV DPTR,#100H    ; LOAD DATA POINTER WITH A DUMMY NUMBER
        MOVX A,@DPTR      ; READ THE COMMAND BYTE.
        ANL A,#00001111B  ; KEEP THE OPERATION FIELD
        MOV R2,A          ; SAVE COMMAND

; DECODE COMMAND AND JUMP TO THE APPROPRIATE ROUTINE
; COMMAND OPERATION (BITS0-3)
;
; ABORT 00H
; REC-DISABLE 01H
; TRA-DISABLE 02H
; RECEIVE 03H
; TRANSMIT 04H
; DUMP 05H
; CONFIGURE 06H
; NOP 07H

        JNB PX0,J1        ; IF INTO IS SET TO PRIORITY 1,
        JMP CABO          ; THEN DMA OPERATION WAS IN PROGRESS.
                          ; EXECUTE ABORT REGARDLESS OF THE
                          ; COMMAND ISSUED.
J1:      CJNE A,#00H,J2
        JMP CABO          ; EXECUTE ABORT
                          ; THIS LINE WILL BE EXECUTED IF ABORT WAS
                          ; ISSUED WHEN THE 8044 IS NOT EXECUTING
                          ; ANY COMMANDS.
J2:      CJNE A,#01H,J3
        JMP CRDIS         ; EXECUTE RECEIVE-DISCONNECT
J3:      CJNE A,#0B5H,J4
        JMP CTDIS         ; EXECUTE TRANSMIT-DISCONNECT
J4:      CJNE A,#03H,J5
        JMP CREC          ; EXECUTE RECEIVE
J5:      CJNE A,#04H,J6
        JMP CTRA          ; EXECUTE TRANSMIT
J6:      CJNE A,#05H,J7
        JMP CDUMP         ; EXECUTE DUMP
J7:      CJNE A,#06H,J8
        JMP CCON          ; EXECUTE CONFIGURE
J8:      CJNE A,#07H,J9
        JMP CNOP          ; EXECUTE NOP
J9:      RETI             ; RETURN. OPERATION NOT RECOGNIZED.

```

231784-23

Figure A-8. External Interrupt Service Routine


```

; ** NOP COMMAND
CNOP:   CLR    IE0                ; IGNORE PENDING EXT INTO (IF ANY).
                                           ; ANY INTERRUPT (COMMAD) DURING
                                           ; EXECUTION OF AN OPERATION IS IGNORED
                                           ; RETURN

      RETI

; ** ABORT COMMAND
CABO:   JNB    PX0,CABOJ1          ; WAS DMA IN PROGRESS?
      CLR    PX0                ; YES. EXT INTO: PRIORITY 0
      CLR    P1.1              ; CLEAR DMA REQUEST

      SETB   P1.2              ; UPDATE STATUS WITH
      SETB   P1.3              ; ABORT-DONE EVENT
      SETB   P1.4              ; (STATUS=DDH; E=0)

      CLR    IE0                ; IGNORE PENDING EXT INTO (IF ANY).
      CLR    P1.0              ; SET INT BIT AND INTERRUPT 80186
      SETB   P1.0              ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
      JB     P3.2,$            ; EXECUTE THE NEXT "RETI" TWICE

      POP    ACC                ; POP OUT THE OLD HI BYTE PC
      POP    ACC                ; POP OUT THE OLD LOW BYTE PC
      MOV    B,#HIGH($+10)      ; HI BYTE ADDRESS OF CABOJ2
      MOV    ACC,#LOW($+7)      ; LOW BYTE ADDRESS OF CABOJ2
      PUSH   ACC                ; PUSH THE ADDRESS OF THE NEXT
      PUSH   B                  ; "RETI" INSTRUCTION INTO STACK
CABOJ2: RETI                    ; RETURN

CABOJ1: NOP                     ; DMA WAS NOT IN PROGRESS
      SETB   P1.5              ; SET THE E BIT

      SETB   P1.2              ; UPDATE STATUS WITH
      SETB   P1.3              ; ABORT-DONE EVENT
      SETB   P1.4              ; (STATUS=FDH; E=1)

      CLR    IE0                ; IGNORE PENDING EXT INTO (IF ANY).
      CLR    P1.0              ; SET INT BIT AND INTERRUPT 80186
      SETB   P1.0              ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
      JB     P3.2,$            ; RETURN
      RETI

```

231784-24

Figure A-9. Execution of NOP and ABORT Commands

```

; ** CONFIGURE COMMAD
CCON:   MOV    DPTR,#100H
      CLR    IE0
      SETB   PX0

      SETB   P1.1              ; IGNORE PENDING EXT INTO (IF ANY)
      JB     P3.3,$            ; EXT INTO: PRIORITY 1
      MOVX   A,@DPTR           ; PX0 IS SET TO ACCEPT ABORT
      MOV    R0,A              ; DURING DMA OPERATION.
      DEC    R0                ; ENABLE DMA REQUEST
      JB     P3.3,$            ; WAIT FOR DMA ACK.
      MOVX   A,@DPTR           ; READ FROM COMMAD/DATA REGISTER
      MOV    R1,A              ; LOAD BYTE COUNT
      JB     P3.3,$            ; DECREMENT BYTE COUNT
      MOVX   A,@DPTR           ; WAIT FOR DMA ACK.
      MOV    R1,A              ; READ FROM COMMAND/DATA REGISTER
      CJNE   R1,#01H,CCONJ1    ; CHECK THE FIRST-BYTE
      MOV    STS,A             ; UPDATE THE STS REGISTER
      INC    R1                ; INC. POINTER TO THE CONF. BLOCK
      DJNZ   R0,CCONF4         ; CHECK THE BYTE COUNT
      JMP    CCONT1

CCONF4: JB     P3.3,CCONF4
      MOVX   A,@DPTR
      MOV    R1,A
      JB     P3.3,$
      MOVX   A,@DPTR
      CJNE   R1,#02H,CCONJ2
      MOV    SMD,A
      INC    R1
      DJNZ   R0,CCONF5
      JMP    CCONT1

CCONF5: JB     P3.3,CCONF5
      MOVX   A,@DPTR
      CJNE   R1,#03H,CCONJ3
      MOV    STAD,A
      INC    R1
      DJNZ   R0,CCONF6
      JMP    CCONT1

CCONF6: JB     P3.3,CCONF6
      MOVX   A,@DPTR
      CJNE   R1,#04H,CCONJ4

```

231784-25

Figure A-10. Execution of CONFIGURE Command

	MOV	TBS,A			
	INC	R1			
	DJNZ	RO,CCONF7			
	JMP	CCONT1			
CCONF7:	JB	P3.3,CCONF7			
	MOVX	A,#DPTR			
CCONJ4:	CJNE	R1,#05H,CCONJ5			
	MOV	TBL,A			
	INC	R1			
	DJNZ	RO,CCONF8			
	JMP	CCONT1			
CCONF8:	JB	P3.3,CCONF8			
	MOVX	A,#DPTR			
CCONJ5:	CJNE	R1,#06H,CCONJ6			
	MOV	RBS,A			
	INC	R1			
	DJNZ	RO,CCONF9			
	JMP	CCONT1			
CCONF9:	JB	P3.3,CCONF9			
	MOVX	A,#DPTR			
CCONJ6:	CJNE	R1,#07H,CCONJ7			
	MOV	RBL,A			
	INC	R1			
	DJNZ	RO,CCONFA			
	JMP	CCONT1			
CCONFA:	JB	P3.3,CCONFA			
	MOVX	A,#DPTR			
CCONJ7:	CJNE	R1,#08H,CCONJ8			
	MOV	IP,A			
	INC	R1			
	DJNZ	RO,CCONF8			
	JMP	CCONT1			
CCONFB:	JB	P3.3,CCONFB			
	MOVX	A,#DPTR			
CCONJ8:	CJNE	R1,#09H,CCONJ9			
	MOV	IE,A			
	INC	R1			
	DJNZ	RO,CCONF8			
	JMP	CCONT1			
CCONFC:	JB	P3.3,CCONFC			
	MOVX	A,#DPTR			
CCONJ9:	CJNE	R1,#0AH,CCONJA			
	MOV	TMOD,A			
	INC	R1			
	DJNZ	RO,CCONF8			
	JMP	CCONT1			
CCONFD:	JB	P3.3,CCONFD			
	MOVX	A,#DPTR			
CCONJA:	CJNE	R1,#0BH,CCONJB			
	MOV	TCON,A			
	INC	R1			
	DJNZ	RO,CCONF8			
	JMP	CCONT1			
CCONFE:	JB	P3.3,CCONFE			
	MOVX	A,#DPTR			
CCONJB:	CJNE	R1,#0CH,ERROR1			
	MOV	PSW,A			
	INC	R1			
	DJNZ	RO,ERROR1			
	JMP	CCONT1			
ERROR1:	NOP				
	SETB	P1.5			
CCONT1:	NOP				
	CLR	P1.1			
	CLR	PX0			
	SETB	P1.2			
	CLR	P1.3			
	CLR	P1.4			
	CLR	IE0			
	CLR	P1.0			
	SETB	P1.0			
	JB	P3.2,\$			
	RETI				

Figure A-10. Execution of CONFIGURE Command (Continued)

```

; ** DUMP COMMAND
CDUMP:  MOV  A,STS      ; LOAD THE FIRST DUMP REG INTO ACC
        MOVX @DPTR,A  ; WRITE TO THE COMMAND/DATA REGISTER
        CLR  IE0       ; IGNORE PENDING EXT INTO (IF ANY)
        SETB PX0       ; INTERRUPT 0: PRIORITY 1
        SETB P1.1      ; ENABLE DMA REQUEST
        JB   P3.3,$     ; WAIT FOR DMA ACK
        MOV  A,SMD
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,STAD
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,TBS
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,TBL
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,TCB
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,RBS
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,RBL
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,RCE
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,RFL
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,PSW
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,IP
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,IE
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,TMOD
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,TCO
        MOVX @DPTR,A
        JB   P3.3,$
        CLR  P1.1      ; DISABLE DRQ
        CLR  PX0       ; EXTERNAL INTO: PRIORITY 0
        SETB P1.2      ; UPDATE STATUS WITH
        SETB P1.3      ; DUMP-DONE EVENT
        CLR  P1.4      ; (STATUS=CDH)
        CLR  IE0       ; IGNORE PENDING EXT INTO
        CLR  P1.0
        SETB P1.0      ; INTERRUPT THE 80186
        JB   P3.2,$     ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
        RETI          ; RETURN

```

231784-28

231784-29

Figure A-11. Execution of DUMP Command

```

; ** RECEIVE COMMAND.
CREC:  JNB  RBE,CRECJ1      ; IS SIU ALREADY IN RECEIVE MODE?
      SETB P1.5            ; YES. SET THE E BIT
CRECJ1: SETB RBE           ; NO. ENABLE RECEPTION
      CLR  RBP             ; CLEAR RECEIVE BUFFER PROTECT BIT
      CLR  IEO             ; IGNORE PENDING EXT INTO (IF ANY)
      RETI                 ; RETURN. UPDATE STATUS IN THE
                          ; SIU INTERRUPT ROUTINE.

; ** TRANSMIT COMMAND.
CTRA:  MOV  R1,TBS         ; LOAD TRANSMIT BUFFER START
      CLR  IEO             ; IGNORE PENDING EXT INTO (IF ANY)
      SETB PX0             ; EXT INTO: PRIORITY 1
      SETB P1.1           ; ENABLE DMA REQUEST
      JB   P3.3,$         ; WAIT FOR DMA ACK.
      MOVX A,@DPTR         ; READ FROM COMMAND/DATA REG.
      MOV  R0,A            ; LOAD THE BYTE COUNT
      DEC  A               ; SUBTRACT 2 FROM THE BYTE
      DEC  A               ; COUNT AND LOAD INTO XMIT
      MOV  TBL,A           ; LOAD BUFFER LENGTH
CTRAJ2: JB   P3.3,CTRAJ2   ; WAIT FOR DMA ACK.
      MOVX A,@DPTR         ; READ FROM COMMAND/DATA REG.
      MOV  STAD,A          ; LOAD DESTINATION ADDRESS
      DEC  R0              ; DECREMENT THE BYTE COUNT
CTRAJ3: JB   P3.3,CTRAJ3   ; WAIT FOR DMA ACK.
      MOVX A,@DPTR         ; READ FROM COMMAND/DATA REG.
      MOV  TCB,A           ; LOAD THE TRANSMIT CONTROL BYTE
      DJNZ R0,CTRAJ4       ; IS THERE ANY INFO. BYTE?
      SJMP CTRAJ5          ; NO.
CTRAJ4: JB   P3.3,CTRAJ4   ; YES. WAIT FOR DMA ACK.
      MOVX A,@DPTR         ; READ FROM COMMAND/DATA REG.
      MOV  @R1,A           ; MOVE DATA TO THE TRANSMIT BUFFER
      INC  R1              ; INC. POINTER TO BUFFER
      DJNZ R0,CTRAJ4       ; LAST BYTE FETCHED INTO THE BUFFER?
      ; NO. FETCH THE NEXT BYTE
CTRAJ5: CLR  P1.1          ; YES. DISABLE DMA REQUEST
      CLR  PX0             ; EXT INTO: PRIORITY 0
      SETB TBF            ; SET TRANSMIT BUFFER FULL
      SETB RTS            ; ENABLE TRANSMISSION
      CLR  IEO             ; IGNORE PENDING EXT INTO (IF ANY)
      RETI                 ; RETURN. UPDATE STATUS IN THE
                          ; SIU INTERRUPT ROUTINE

```

231784-30

Figure A-12. Execution of RECEIVE and TRANSMIT Commands

```

; ** TRANSMIT-DISCONNECT COMMAND
CTDIS: JB   TBF,CTDIJ1     ; IS TRANSMIT BUFFER ALREADY EMPTY?
      SETB P1.5            ; YES, SET THE E BIT
CTDIJ1: CLR  TBF           ; NO. CLEAR TRANSMIT BUFFER
      CLR  IEO             ; IGNORE PENDING EXT INTO (IF ANY)
      RETI                 ; RETURN. UPDATE STATUS IN THE
                          ; SIU INTERRUPT ROUTINE.

; ** RECEIVE-DISCONNECT COMMAND
CRDIS: JB   RBE,CRDIJ1     ; IS RECEIVE BUFFER ALREADY EMPTY?
      SETB P1.5            ; YES. SET THE E BIT
CRDIJ1: CLR  RBE           ; NO. CLEAR RECEIVE BUFFER

      SETB P1.2            ; UPDATE STATUS WITH
      CLR  P1.3            ; RECEPTION-DISABLED EVENT
      SETB P1.4            ; (STATUS=D5 IF E=0)

      CLR  IEO             ; INTERRUPT THE 80186
      CLR  P1.0            ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
      SETB P1.0            ;
      JB   P3.2,$         ;
      RETI                 ; RETURN

```

231784-31

Figure A-13. Execution of RECEIVE-DISCONNECT and TRANSMIT-DISCONNECT Commands


```

;***** SERIAL CHANNEL (SIU) INTERRUPT *****
SIINT:  CLR    SI          ; LOAD THE OPERATION FIELD
        MOV    A,R2        ; RECEIVE COMMAND PENDING?
        CJNE   A,#03H,SINTJ1
        JMP     SIREC        ; YES.
SINTJ1:  CJNE   A,#02H,SINTJ2 ; TRANSMIT-DISCONNECT PENDING?
        JMP     SITDIS        ; YES.
SINTJ2:  JMP     SITRA        ; TRANSMIT COMMAND IS PENDING

; ** TRANSMISSION IS DISABLED
SITDIS:  JB     RTS,SINTJ3    ; REQUEST TO SEND ENABLED?
        JNB    TBF,SINTJ3    ; YES. TRANSMISSION DISABLED?
        ; YES.
        CLR    P1.2          ; UPDATE STATUS WITH
        SETB   P1.3          ; TRANSMISSION-DISABLED EVENT
        SETB   P1.4          ; (STATUS=D9H)

        CLR    IE0          ; IGNORE PENDING EXT INTO
        CLR    P1.0
        SETB   P1.0          ; INTERRUPT THE 80186
        JB     P3.2,$        ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
        RETI

; ** A FRAME IS TRANSMITTED
SITRA:   JB     RTS,SINTJ3    ; A FRAME TRANSMITTED?
        ; YES.
        CLR    P1.2          ; UPDATE STATUS WITH
        SETB   P1.3          ; TRANSMIT-DONE EVENT
        SETB   P1.4          ; (STATUS=C9).

        CLR    IE0
        CLR    P1.0
        SETB   P1.0          ; INTERRUPT THE 80186
        JB     P3.2,$        ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
        RETI

; ** A FRAME IS RECEIVED
SIREC:   JB     RBE,SINTJ3    ; RECEIVE BUFFER FULL?
        JNB    BOV,SINTJ4    ; YES. BUFFER OVERRUN?
        SETB   P1.5          ; YES. SET THE E BIT
SINTJ4:   MOV    R0,RFL        ; LOAD R0 WITH RECEIVE BYTE COUNT
        MOV    R1,RBS        ; LOAD R1 WITH RECEIVE BUFFER ADDRESS
        CLR    IE0          ; IGNORE PENDING EXT INTO (IF ANY)
        SETB   PX0          ; EXT INTO: PRIORITY 1

        MOV    A,@R1          ; MOVE FIRST BYTE INTO ACC.
        MOVX   @DPTR,A        ; WRITE TO THE COMMAND/DATA REG
        SETB   P1.1          ; ENABLE DMA REQUEST
        INC    R1            ; INC POINTER TO RECEIVE BUFFER
        JB     P3.3,$        ; WAIT FOR DMA ACK.
        DJNZ   RO,CINTJ7      ; LAST BYTE MOVED?
        SJMP   CINTJ8        ; YES

CINTJ7:   MOV    A,@R1          ; LOAD RECEIVED DATA INTO ACC.
        MOVX   @DPTR,A        ; WRITE TO THE COMMAND/DATA REG.
        INC    R1            ; INC POINTER TO RECEIVE BUFFER
        JB     P3.3,$        ; WAIT TILL DMA ACK
        DJNZ   RO,CINTJ7      ; LAST BYTE MOVED TO COMMAND/DATA REG?
        ; NO. DEPOSIT THE NEXT BYTE
        ; LOAD BYTE COUNT
        MOV    A,RFL          ; WRITE TO THE COMMAND/DATA REG
        MOVX   @DPTR,A        ; WAIT FOR DMA ACK.
        JB     P3.3,$        ; LOAD STATION ADDRESS
        MOV    A,STAD         ; WRITE TO THE COMMAND/DATA REG
        MOVX   @DPTR,A        ; WAIT FOR DMA ACK.
        JB     P3.3,$        ; LOAD RECEIVE CONTROL BYTE
        MOVX   @DPTR,A        ; WRITE TO THE COMMAND/DATA REG
        JB     P3.3,$        ; WAIT FOR DMA ACK.
        CLR    P1.1          ; CLEAR DMA REQUEST
        CLR    PX0          ; EXTERNAL INTERRUPT: PRIORITY 0

```

231784-32

231784-33

Figure A-14. Serial Channel Interrupt Routine

231784-34

April 1989

80186/80188 DMA Latency

5

STEVE FARRER
APPLICATIONS ENGINEER

Order Number: 270525-001

**80186/80188 DMA
LATENCY****CONTENTS**

PAGE

DMA REQUEST GENERATION 5-53

Conditions Affecting DMA Latency 5-53

April 1989

80186/80188 DMA Latency

STEVE FARRER
APPLICATIONS ENGINEER

When using the DMA controller of the 80186 and 80188, there are several operating conditions which affect the service time (latency) between when the DMA request is generated and when the bus cycles associated to the DMA transfer are actually run. This application brief describes those conditions which affect DMA Latency.

DMA REQUEST GENERATION

The minimum DMA latency is 4 clocks and, depending on when the signal arrives (i.e. if the signal just missed the setup time), it might appear to be almost 5 clocks. This 4 to 5 clock delay is due to a two phase synchronizer and various transfer gate delays the DRQ signal must take before reaching the BIU. Conceptually the circuit looks like Figure 1.

If the Bus Interface Unit (BIU) is available when the DRQ signal reaches it, then a DMA cycle will proceed at T1 of the bus cycle as the next clock.

Also note that the DRQ signal is not latched, and must remain active until serviced. If the DRQ signal is brought low after being asserted high, then a '0' will propagate through and; if the request had not yet been serviced, then the BIU will see a '0' and the cycle will never take place.

Conditions Affecting DMA Latency

The circumstances that affect DMA latency in order of worst case are as follows:

- 1) HOLD
- 2) LOCK - INTA
- 3) Odd byte accesses
- 4) Effective Address Calculations (EA)

HOLD can indefinitely delay a DMA cycle. There is no mechanism internally to remove HLDA when a DMA request is pending.

LOCKed instructions can also delay a DMA cycle by a significant amount, depending on the type of instruction locked. A typical locked XCHG instruction from memory to register could delay the DMA cycle by as much as 18 clocks if the memory access required two bus cycles (80188 or odd locations on the 80186). On the other hand, a locked repeat MOVS could delay a DMA cycle by up to 1.05 million clocks depending on the number of transfers and the number of bus cycles per transfers.

Interrupt acknowledges can also affect DMA latency because the bus is locked out during the first two bus cycles required to fetch the interrupt vector type. This causes the worst case latency during interrupt acknowledges to be:

4	Clocks (Minimum Setup)
10	Clocks (2 Bus Cycles + 2 Idle Clocks) Min
14	Clocks Total

Both HOLD and LOCK are extremely dependent on the type of system being designed and therefore are not really considered to be normal worst case latency. However, odd byte accesses and effective address calculations are conditions that frequently occur in almost all systems. Under these conditions of no HOLD, no LOCK, and no wait states, the worst case occurs when the DMA request loses to an instruction data cycle requiring an effective address calculation.

Effective addresses (EA) always require 4 clocks for calculation and can only take place during T3-T4-TI-TI, T4-TI-TI-TI, or TI-TI-TI-TI. This creates an extra minimum insertion of 2 T-idle cycles. If the EA requires an immediate value in the prefetch queue, then a signal goes active which places the EA bus cycle at a higher priority than any other BIU requests. This is so the execution unit won't be waiting on the bus interface unit. If the EA hadn't required the value in the queue, then the EU could proceed with the next instruction shortly after it had sent the request to the BIU. Figure 2 shows the effects EA calculations have on DMA Latency.

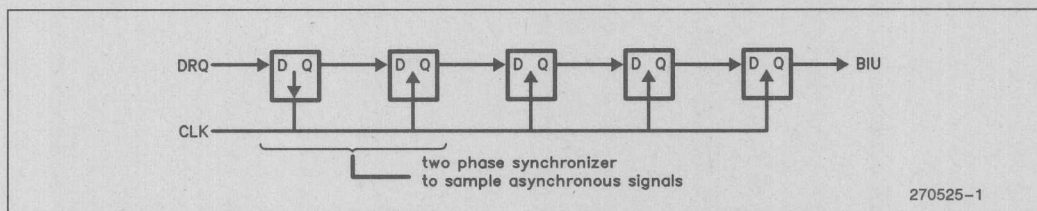


Figure 1. DMA Request Synchronization

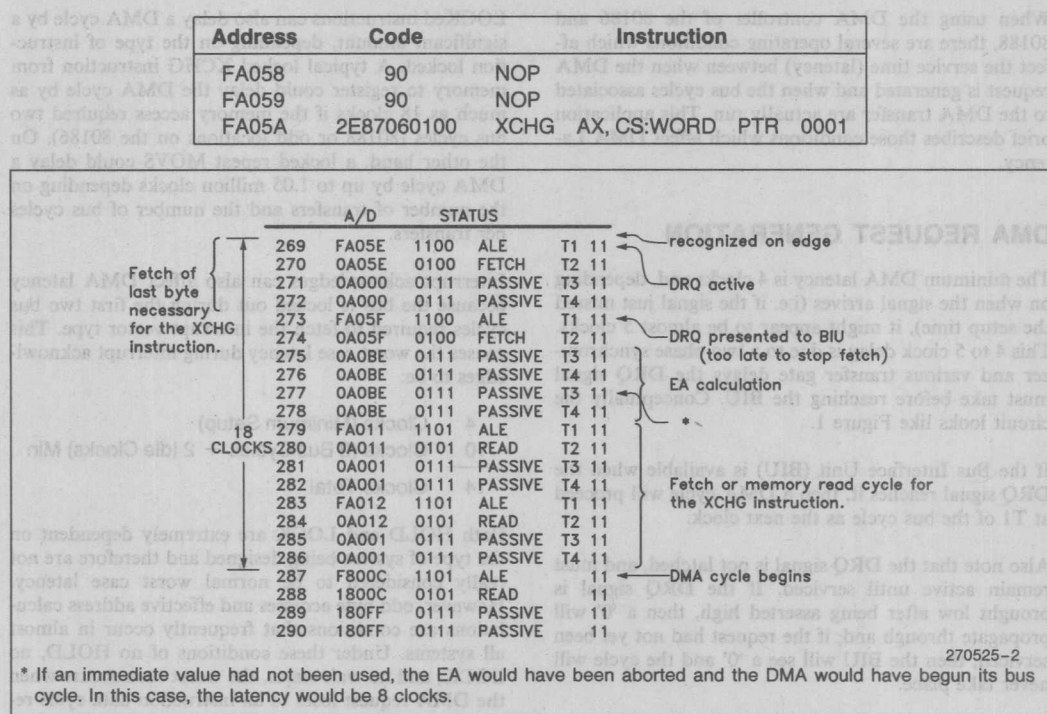


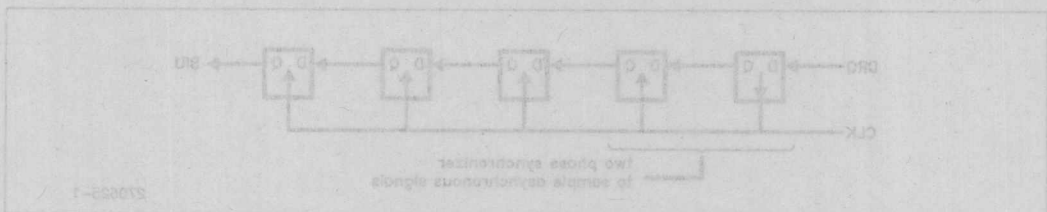
Figure 2. Logic State Analyzer Trace and Accompanying Program Code

Effective addresses (EA) always require 4 clocks for calculation and can only take place during T3-T4-T1-T2 or T1-T2-T3-T4. This requires an extra minimum insertion of 2 T-tile cycles. If the EA requires an immediate value in the previous cycle, then a signal goes active which places the EA bus cycle at a higher priority than any other BIU request. This is so the execution unit won't be waiting on values in the queue, then the EA could proceed with the next instruction shortly after it had sent the request to the BIU. Figure 2 shows the effective EA calculations have on DMA latency.

The circumstances that affect DMA latency in order of worst case are as follows:

- 1) HOLD
- 2) LOCK - DATA
- 3) Odd byte addresses
- 4) Effective Address Calculations (EA)

HOLD can indefinitely delay a DMA cycle. There is no mechanism internally to remove HLLDA when a DMA request is pending.



April 1989

80186/80188 EFI Drive and Oscillator Operation

5

STEVE FARRER
APPLICATIONS ENGINEER

Order Number: 270526-001

80186/80188 EFI DRIVE AND OSCILLATOR OPERATION

CONTENTS

PAGE

EFI Operation 5-57

Crystal Operation 5-57

April 1989

80186/80188 EFI Drive and Oscillator Operation

STEVE FARRER
APPLICATIONS ENGINEER

Order Number: 570528-001

There has been some confusion in the past regarding the correct input for EFI (External Frequency Input) use and what parameters should be used for crystal selection. This Application Brief discusses the trade-offs with each input so that one can decide which input suits his design and also lists the parameters for crystal selection.

EFI Operation

The oscillator circuit on the 186/188 is as shown in Figure 1 (simplified). Either input may be used for an EFI signal. Using X1 requires very little drive from an external oscillator since it is essentially the gate of an NMOS transistor. Clock operation works fine using this input, but at higher frequencies the stray capacitance on X2 begins to change the duty cycle of the clock. This will eventually cause the part to fail.

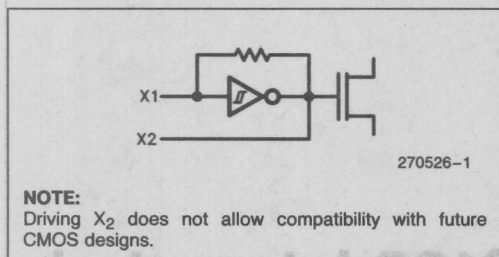


Figure 1. Oscillator Circuit on the 186/188

Using X2 as an EFI gives a broader frequency range but places a more stringent requirement on the drive

capability of the external oscillator. Since X1 is an input, it may be grounded to minimize the capacitance. This in turn allows for a higher frequency range since the duty cycle remains closer to 50%. But with X1 grounded, the output of the inverter (which is directly connected to X2) is always trying to output a high. This means the oscillator driving X2 must be capable of sinking up to 15 mA at cold temperatures when trying to drive it low. If the external oscillator is capable of supplying 15 mA, then this method is preferred. Otherwise, X1 should be used as an EFI.

Caution: using X2 for EFI does not allow for CMOS compatibility at a future date.

Crystal Operation

The oscillator circuit is a single stage amplifier connected as a Pierce oscillator. There are no passive components in the oscillator circuit, only a unique combination of depletion and enhancement mode FET's. Characterization of the oscillator circuit showed that operation was optimum with crystal parameters as follows:

ESR (Equivalent Series Resistance)	30 ohms maximum
Co (Shunt Capacitance)	7.0 pf max.
C1 (Load Capacity)	20 pf \pm 2 pf
Drive Level	1 mW max.

This characterization data was supplied by:

Standard Crystal Corporation
9940 East Baldwin Place
El Monte, CA 91731
(213) 443-2121

This in turn allows for a higher frequency rate. The duty cycle remains close to 50%. But with X1 grounded, the output of the inverter (which is directly connected to X2) is always trying to output a high. This means the oscillator driving X1 must be capable of sinking up to 12 mA at cold temperatures when trying to drive it low. If the external oscillator is capable of supplying 12 mA, then this method is preferred. Otherwise, X1 should be used as an EFT.

Caution: using X2 for EFT does not allow for CMOS compatibility at a future date.

Crystal Operation

The oscillator circuit is a single stage amplifier connected as a Pierce oscillator. There are no passive components in the oscillator circuit, only a single component, the crystal. Using X1 requires very little drive from an external oscillator since it is essentially the gate of an NMOS transistor. Clock operation works the same way. This will eventually cause the part to fail.

NOTE: Driving X2 does not allow compatibility with future CMOS designs.

Drive Level: 1 mW max
 C1 (Load Capacitance): 50 pf ± 5 pf
 Co (Shunt Capacitance): 7.0 pf max
 BSR (Equivalent Series Resistance): 30 ohms maximum

use and what parameters should be used for crystal selection. This Application Brief discusses the trade-offs with each input so that one can decide which input suits his design and also lists the parameters for crystal selection.

EFT Operation

The oscillator circuit on the 186/188 is as shown in Figure 1 (simplified). Either input may be used for an EFT signal. Using X1 requires very little drive from an external oscillator since it is essentially the gate of an NMOS transistor. Clock operation works the same way. This will eventually cause the part to fail.

August 1990



The 80C186/80C188 Integrated Refresh Control Unit

GARRY MION
 ECO SENIOR APPLICATIONS ENGINEER

Order Number: 270520-002

THE 80C186/80C188 INTEGRATED REFRESH CONTROL UNIT

This application brief is not intended to be a discussion of dynamic memory controller design. Instead, it will concentrate on the operation of the Refresh Control Unit of the 80C186, and how it can help simplify a memory controller.

The discussions on the following pages apply to BOTH the 80C186 and 80C188 except where noted.

The discussions on the following pages apply to BOTH the 80C186 and 80C188 except where noted.

UNDERSTANDING DYNAMIC MEMORY

Before explaining how memory refreshing is accomplished, some understanding of a Dynamic Random Access Memory (DRAM) device is needed. Figure 1 shows a simplified block diagram of a DRAM device while a block diagram of a typical dynamic memory controller is shown in Figure 2.

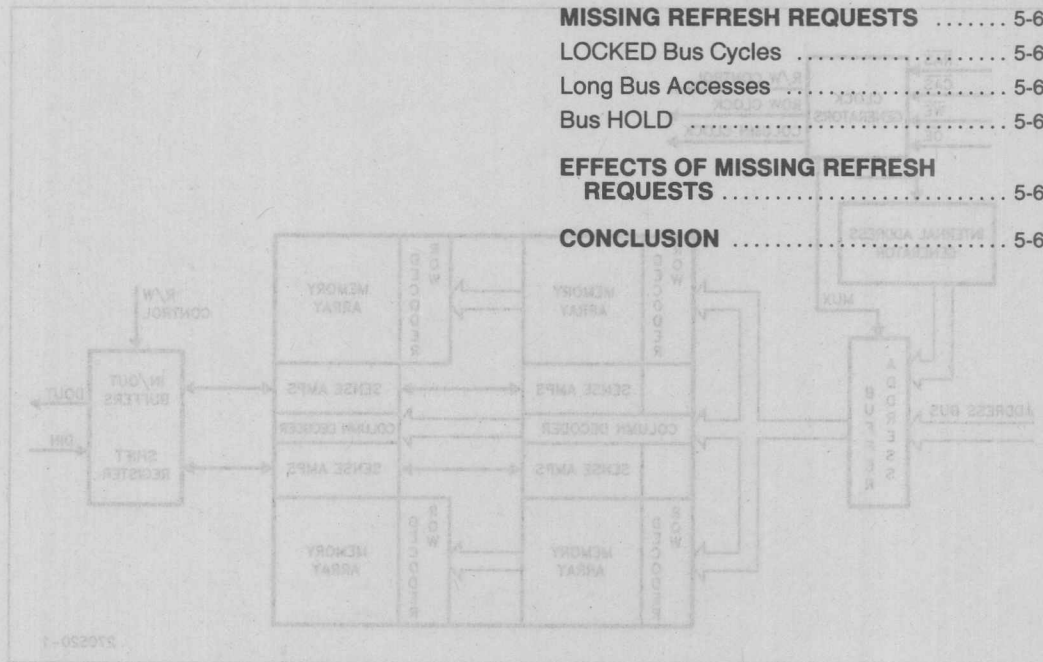


Figure 1. Random Access Memory Device

CONTENTS	PAGE
UNDERSTANDING DYNAMIC MEMORY	5-60
UNDERSTANDING MEMORY REFRESH	5-62
WAYS TO REFRESH A MEMORY DEVICE	5-62
80C186 REFRESH CONTROL FEATURES	5-64
PROGRAMMING CHARACTERISTICS OF THE REFRESH CONTROL UNIT	5-64
Programming the Memory Partition Register	5-65
Programming the Refresh Clock Register	5-66
Programming the Refresh Enable Register	5-67
REFRESH CONTROL UNIT OPERATION	5-67
80C188 Address Considerations	5-68
MISSING REFRESH REQUESTS	5-68
LOCKED Bus Cycles	5-68
Long Bus Accesses	5-69
Bus HOLD	5-69
EFFECTS OF MISSING REFRESH REQUESTS	5-69
CONCLUSION	5-69

The 80C186 and 80C188 incorporate a special control unit that integrates address and clock counters which, along with the Bus Interface Unit (BIU), facilitates dynamic memory refreshing. Refreshing is an operation required by dynamic memory to ensure data retention.

Dynamic memory refreshing can be controlled using anything from an exotic memory controller to a simple timer along with a DMA controller. In fact, the 80186 device accomplishes the task memory refreshing by using one of the internal timer/counters and a DMA channel. However, doing this meant that desirable internal functions were no longer available to do more useful work.

Dynamic memory, unlike static or non-volatile memory, always require some form of a memory controller to enable read and write operations. Therefore, even the most basic dynamic memory interface has a minimum set of support logic. The advent of programmable logic and highly integrated dynamic memory has made the job of designing a memory controller somewhat straightforward. However, directly supporting memory refresh can still complicate many controller designs.

The designer of a memory controller must take into account CPU-versus-refresh arbitration and must provide a mechanism to generate periodic refresh requests. Most dynamic memory devices now contain internal

refresh address counters which eliminate the need for external refresh address generation. However, such devices tend to complicate a memory controller design. The 80C186 simplifies dynamic memory controller design by integrating a refresh mechanism into the operation of the CPU.

This application brief is not intended to be a discussion of dynamic memory controller design. Instead, it will concentrate on the operation of the Refresh Control Unit of the 80C186, and how it can help simplify a memory controller.

The discussions on the following pages apply to BOTH the 80C186 and 80C188 except where noted.

UNDERSTANDING DYNAMIC MEMORY

Before explaining how memory refreshing is accomplished, some understanding of a Dynamic Random Access Memory (DRAM) device is needed. Figure 1 shows a simplified block diagram of a DRAM device, while a block diagram of a typical dynamic memory controller is shown in Figure 2.

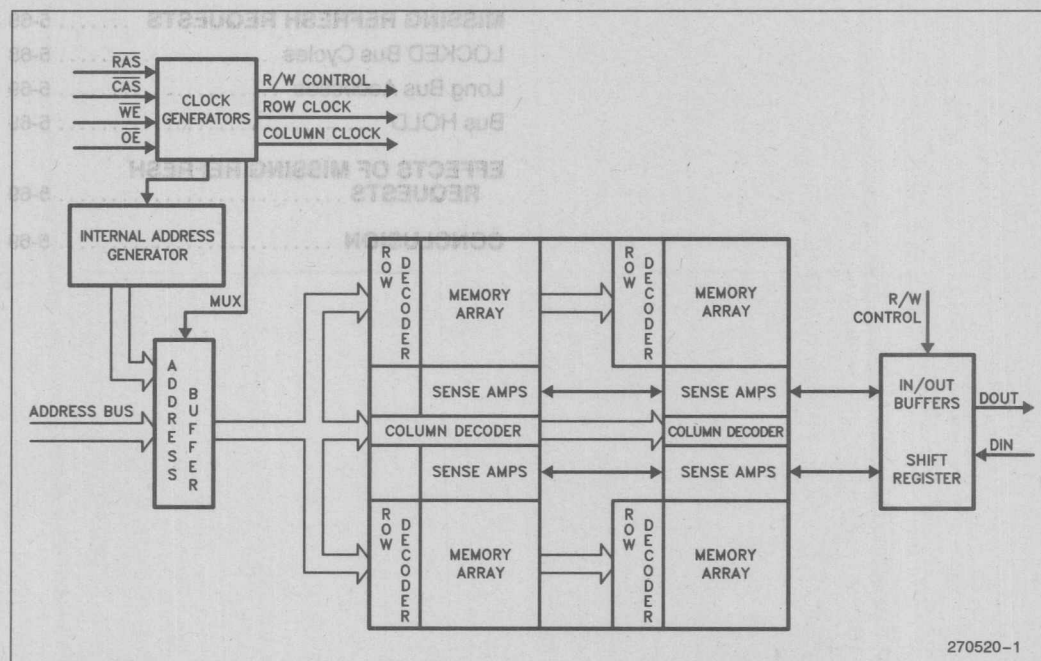


Figure 1. Random Access Memory Device

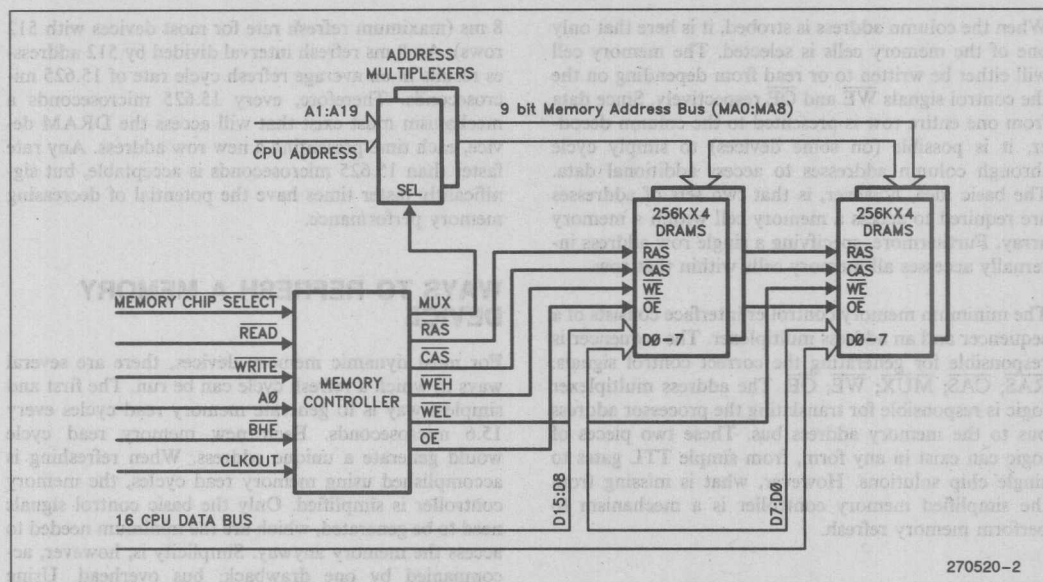


Figure 2. Minimum Configuration Memory Controller

The typical DRAM memory array is built as a matrix. Any bit or cell in the memory array is accessed by specifying a unique row and column address. As shown in Figure 1, the row and column addresses are multiplexed through one set of address inputs. Multiplexing the address inputs helps reduce the number of pins required to support large memory arrays. For instance, adding only one address bit will result in a memory array 4 times as large.

Two control lines, $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$, are used to strobe an address into the memory chip. Figure 3 illustrates a

timing diagram for a typical memory read access and the relationship between the $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ signals. The signal MUX controls which half of the address is presented to the memory devices. After generating the row address strobe ($\overline{\text{RAS}}$), the decoder selects a row of memory cells whose data value will be detected by a Sense Amplifier. The Sense Amplifier then presents the data to the column decoder. **Note that all cells associated to a row get accessed.** The fact that all cells within a row are accessed will be used later to explain why only the $\overline{\text{RAS}}$ portion of the memory address is required to refresh a device.

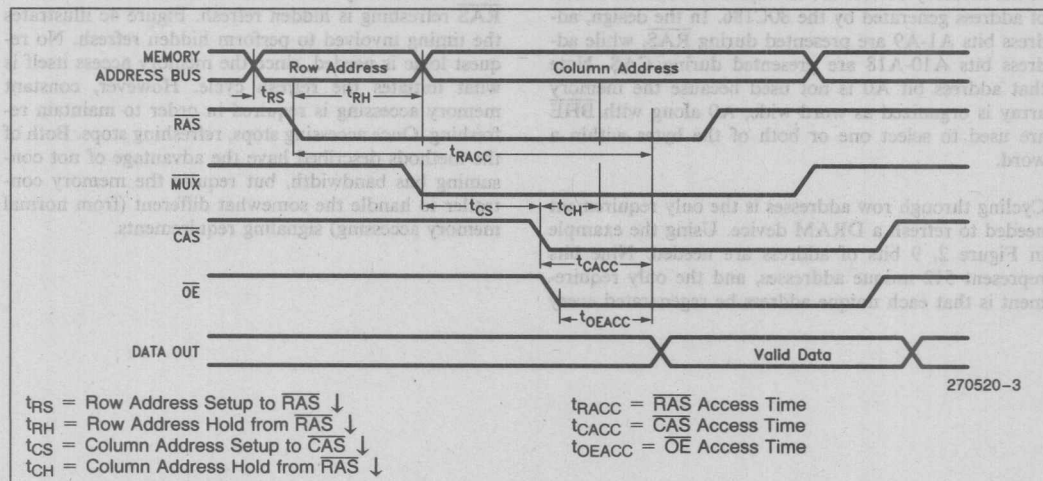


Figure 3. DRAM Signal Timings

When the column address is strobed, it is here that only one of the memory cells is selected. The memory cell will either be written to or read from depending on the the control signals \overline{WE} and \overline{OE} respectively. Since data from one entire row is presented to the column decoder, it is possible (on some devices) to simply cycle through column addresses to access additional data. The basic idea, however, is that two sets of addresses are required to access a memory cell within a memory array. Furthermore, specifying a single row address internally accesses all memory cells within that row.

The minimum memory controller interface consists of a sequencer and an address multiplexer. The sequencer is responsible for generating the correct control signals: \overline{RAS} ; \overline{CAS} ; \overline{MUX} ; \overline{WE} ; \overline{OE} . The address multiplexer logic is responsible for translating the processor address bus to the memory address bus. These two pieces of logic can exist in any form, from simple TTL gates to single chip solutions. However, what is missing from the simplified memory controller is a mechanism to perform memory refresh.

UNDERSTANDING MEMORY REFRESH

As indicated earlier, dynamic memory needs to be refreshed in order to maintain its data. Refreshing is accomplished whenever a memory cell is accessed. It is not necessary to read a memory location and then write the value back in order to refresh a memory cell. Simply cycling through a complete set of row addresses is all that is required. Remember, since a row accesses all memory cells associated to it, accessing all rows will access all the cells within the device.

Referring back to Figure 2, the 9 address bits presented to the memory devices are multiplexed from the 18 bits of address generated by the 80C186. In the design, address bits A1-A9 are presented during \overline{RAS} , while address bits A10-A18 are presented during \overline{CAS} . Note that address bit A0 is not used because the memory array is organized as word wide; A0 along with \overline{BHE} are used to select one or both of the bytes within a word.

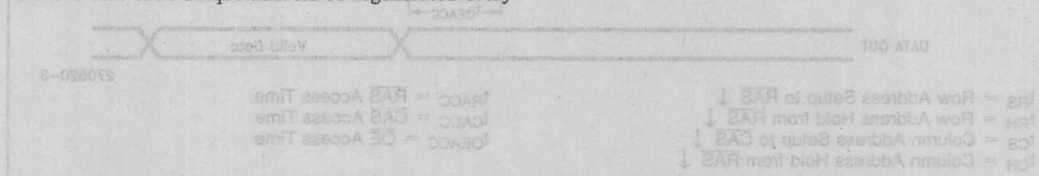
Cycling through row addresses is the only requirement needed to refresh a DRAM device. Using the example in Figure 2, 9 bits of address are needed. Nine bits represent 512 unique addresses, and the only requirement is that each unique address be regenerated every

8 ms (maximum refresh rate for most devices with 512 rows). An 8 ms refresh interval divided by 512 addresses results in an average refresh cycle rate of 15.625 microseconds. Therefore, every 15.625 microseconds a mechanism must exist that will access the DRAM device, each time presenting a new row address. Any rate faster than 15.625 microseconds is acceptable, but significantly faster times have the potential of decreasing memory performance.

WAYS TO REFRESH A MEMORY DEVICE

For most dynamic memory devices, there are several ways in which a refresh cycle can be run. The first and simplest way is to generate memory read cycles every 15.6 microseconds. Each new memory read cycle would generate a unique address. When refreshing is accomplished using memory read cycles, the memory controller is simplified. Only the basic control signals need to be generated, which are the minimum needed to access the memory anyway. Simplicity is, however, accompanied by one drawback; bus overhead. Using memory reads to perform DRAM refreshing means that one bus cycle every 15.6 microseconds is wasted. When operating at very slow speeds, a wasted bus cycle might appear to be significant. But if a bus cycle takes only, say, 320 nanoseconds to complete, running a refresh cycle every 15.6 microseconds represents a two percent hit in bus performance.

A second method relies on the fact that most dynamic memory devices now have built in refresh address mechanisms. DRAM refreshing can be accomplished by generating \overline{CAS} before \overline{RAS} signaling (see Figure 4a). This method requires that an external signal generate a periodic request to the DRAM controller to initiate the refresh cycle. A method similar to \overline{CAS} before \overline{RAS} refreshing is hidden refresh. Figure 4c illustrates the timing involved to perform hidden refresh. No request logic is needed, since the memory access itself is what initiates the refresh cycle. However, constant memory accessing is required in order to maintain refreshing. Once accessing stops, refreshing stops. Both of the methods described have the advantage of not consuming bus bandwidth, but require the memory controller to handle the somewhat different (from normal memory accessing) signaling requirements.



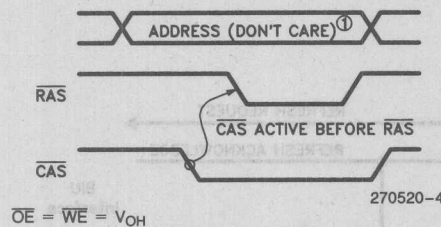


Figure 4a. CAS before RAS Refresh

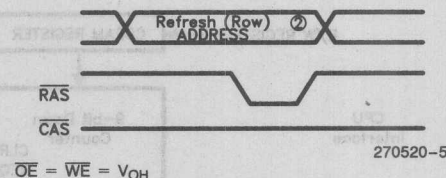


Figure 4b. RAS Only Refresh

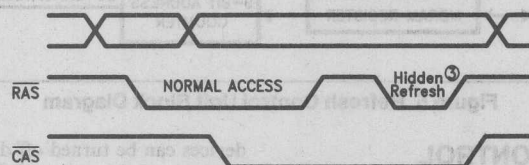


Figure 4c.

NOTES:

1. Refresh address provided internal to memory device.
2. Refresh address presented external to memory device.
3. Refresh address generated internally, and cycle does not effect memory access in progress (i.e. hidden).

Figure 4. Alternate Refreshing Methods

A final method is to implement a discrete design that supports refresh control and refresh address generation. The circuit details are shown in Figure 4b. A discrete design allows the most design flexibility and can be tailored to meet any system-to-memory interfacing requirements.

There are other methods available, most of which involve single-chip dedicated memory controllers. However, any memory controller design that performs the function of refreshing either directly or through external support circuitry has one major concern; arbitration between the refresh cycle and a normal memory access. The best way to make the operation of the DRAM memory controller a true slave to the operation of the

CPU is to include refreshing as part of the functionality of the CPU. By offloading the task of memory refreshing onto the CPU, the memory controller can be simplified and dedicated to the duty of DRAM interfacing.

The idea that the 80C186 refresh cycle is simply a memory read means that the dynamic memory control logic does not need to differentiate between refresh cycles and normal memory read cycles. This simplifies the design of the memory controller. There are no special signaling requirements needed, and RAS only refreshing (for low-power designs) can be easily accommodated. Further, since the request is generated internally and synchronous with the operation of the BIU, no special external logic needs to detect when a refresh cycle conflicts with a CPU access.

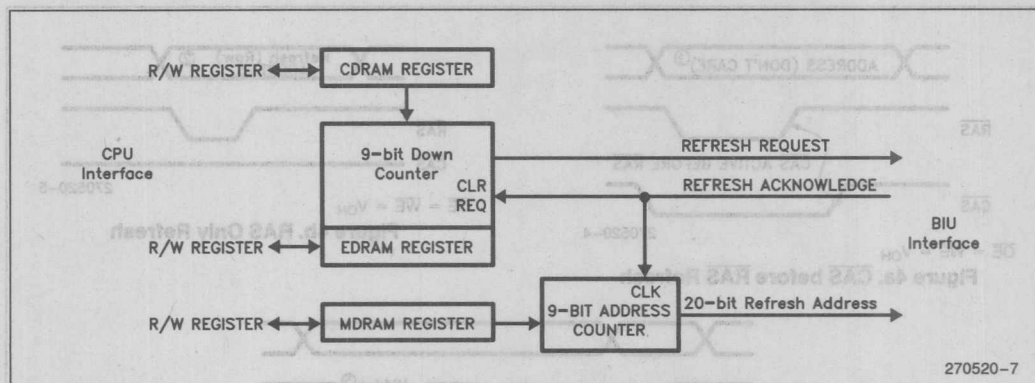


Figure 5. Refresh Control Unit Block Diagram

80C186 REFRESH CONTROL FEATURES

The Refresh Control Unit (RCU) of the 80C186 consists of a 9-bit address counter, a 9-bit down counter, and support logic. The block diagram can be seen in Figure 5.

The 9-bit address counter is controlled by the BIU and used whenever a refresh bus cycle is executed. Thus, any dynamic memory device whose refresh address requirement does not exceed nine bits can be directly supported by the 80C186. A special register has been defined to allow the base (starting) address of the refresh memory region to be specified. This base address can be located on any 4 kilobyte boundary. Furthermore, if this refresh base address overlaps any of the defined chip select regions, the chip select defined for that region will go active.

The 9-bit down counter initiates a refresh request. When the counter decrements to 1 (it decrements every clock cycle), a refresh request is presented to the BIU. When the bus is free, the BIU will run the refresh (memory) bus cycle. Note that since a refresh bus cycle is executed by the BIU, the faster refresh cycles are requested the greater the impact on bus performance. Referring back to the discussion of request rates, the maximum refresh period is typically 15.6 microseconds. With the 80C186 operating at 12.5 MHz, this represents a refresh bus impact of only 2%. However, at 5 microseconds the bus impact is 15%. Therefore, the refresh request rate should be tailored to meet the needs of the dynamic memory and the system. The 80C186 provides flexibility by allowing the request rate to be programmable in 80 ns steps (at 12.5 MHz).

To facilitate low power designs, the refresh bus cycle provides a mechanism whereby the dynamic memory

devices can be turned off during refresh accesses. Low power control is accomplished by driving both address bit A0 and the control signal \overline{BHE} to a high level. Essentially an invalid bus access condition exists, since A0 and \overline{BHE} are used to indicate which half of the data bus is being accessed. When both are high during the access, the indication is that neither half of the bus is being used for the data transfer. This is acceptable for refresh bus cycles since no data is actually being transferred. If the memory controller takes advantage of this condition, the output enables of the dynamic memory devices (as well as the \overline{CAS} strobe) can be disabled during refresh bus cycles, providing overall lower power consumption.

PROGRAMMING CHARACTERISTICS OF THE REFRESH CONTROL UNIT

A block of control registers are defined in the Peripheral Control Block (PCB) that define the operating characteristics of the refresh control unit (refer to Figure 5). These registers are only accessible when the 80C186 is operating in enhanced mode. When in compatibility mode, the 80C186 will ignore any reads or writes to the RCU registers.

The three registers associated with the refresh unit (MDRAM, CDRAM, EDRAM) provide the following features:

- 1) Enable/disable refresh unit
- 2) Establish a refresh request rate
- 3) Establish a refresh memory region
- 4) Examine the refresh down counter

It is not necessary to program any of these registers in a specific sequence, although the refresh request rate and refresh base address registers should be programmed before the refresh unit is enabled.

Programming the Memory Partition Register

The MDRAM register (Figure 6) is used to define address bits A13 through A19 of the 20 bit refresh address. This essentially establishes a memory region which will be accessed during refresh bus cycles. Typically, the refresh memory region will overlap a chip select that is used to access the dynamic memory. Overlapping the refresh memory region with a chip select memory region, means no additional external hardware is needed to support refresh bus cycles since it essentially operates the same as memory read cycles. When the 80C186 is reset, the MDRAM register is initialized to zero.

Figure 7 illustrates how the refresh address is generated. Address bits A10-A12 are not programmable and are always driven to a zero during a refresh bus cycle. Address bits A1 through A9 are derived by a 9-bit linear-feedback shift counter. The address counter is not ascending or contiguous, meaning that the counter does not start at 0 and increment to 511 before resetting back to 0. For refreshing purposes, it is not important that the address be contiguous and count up or down. Rather, the only requirement is that all combinations of the 512 addresses be cycled through before being repeated. Equation 1 provides the state definition of the 9-bit refresh address counter and can be used to determine the exact counting sequence. Figure 8 illustrates the gate logic used to create such a counter.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MDRAM:	M6	M5	M4	M3	M2	M1	M0	0	0	0	0	0	0	0	0	0

bits 0-8: Reserved, should be written as a 0 to maintain future compatibility, will be read back as 0.
bits 9-15: M0-M6, are used to define address bits A13-A19 (respectively) of the 20-bit memory refresh address. These bits are set to 0 on RESET.

Figure 6. MDRAM Register Format

Address Bit	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Refresh Address	M6	M5	M4	M3	M2	M1	M0	0	0	0	C _{A8}	C _{A7}	C _{A6}	C _{A5}	C _{A4}	C _{A3}	C _{A2}	C _{A1}	C _{A0}	1

Bit 0: Always driven to a 1 (Logic High). This is true for both the 80C186/80C188.
Bits 1-9: C_{A0}-C_{A8}, are generated by the 9-bit Linear-Feedback shift counter.
Bits 10-12: Always driven to a 0 (Logic Low).
Bits 13-19: M0-M6, are defined by the MDRAM Register.

Figure 7. Physical Refresh Address Generation

$$\begin{aligned}
 C_{A0} &\leftarrow C_{A1} \\
 C_{A1} &\leftarrow C_{A2} \\
 C_{A2} &\leftarrow C_{A3} \\
 C_{A3} &\leftarrow C_{A4} \\
 C_{A4} &\leftarrow C_{A5} \\
 C_{A5} &\leftarrow C_{A6} \\
 C_{A6} &\leftarrow \text{If } (C_{A1}-C_{A6} = 1111111B), \\
 &\quad \text{then } C_{A6} = \text{Inverted } C_{A0} \\
 &\quad \text{else } C_{A6} = ((C_{A0} \cdot C_{A1}) \cdot \text{XNOR} \cdot (C_{A2} \cdot C_{A3})) \\
 C_{A7}, C_{A8} &\leftarrow \text{If } (C_{A0} - C_{A6} = 0111111B) \\
 &\quad \text{then } C_{A7}, C_{A8} = C_{A7}, C_{A8} + 1 \\
 &\quad \text{else } C_{A7}, C_{A8} = C_{A7}, C_{A8}
 \end{aligned}$$

Equation 1. Refresh Counter Operation

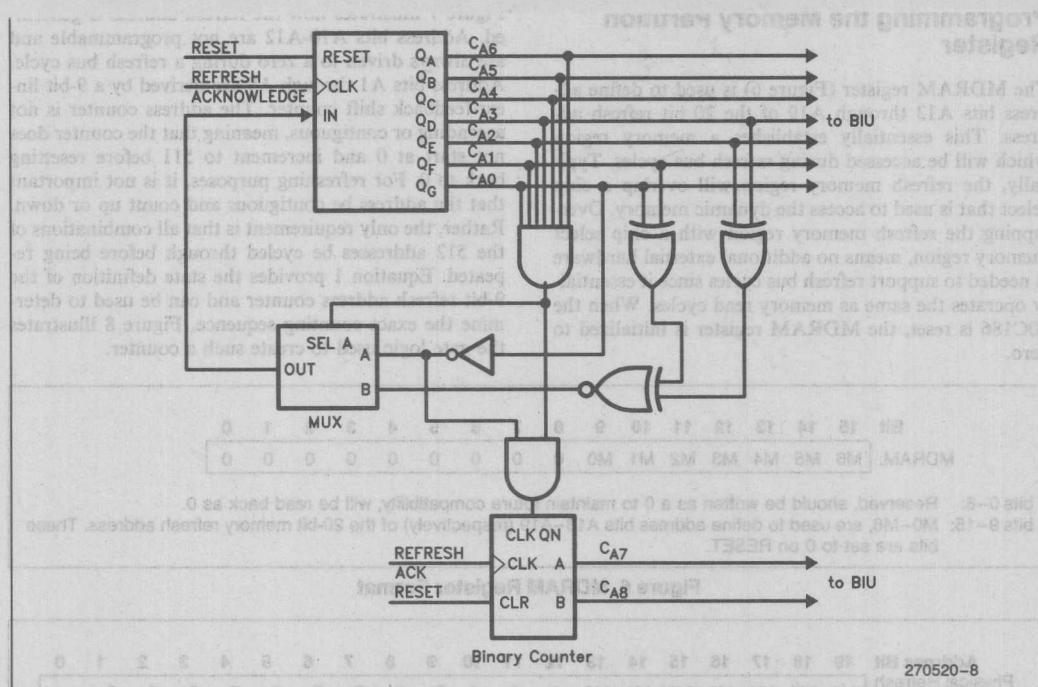


Figure 8. Logic Representation of Refresh Address Counter

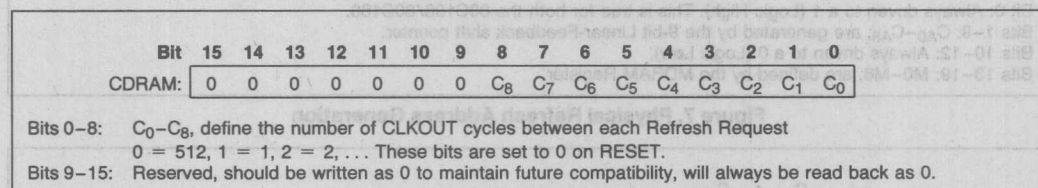


Figure 9. CDRAM Register Format

There are no limitations placed on the programming of the MDRAM register, but be aware that any chip select **memory** region that overlaps the address established by the MDRAM register will be activated during refresh bus cycles. Therefore, the register should be programmed to correspond to the chip select address that is activated for the dynamic memory partition.

Programming the Refresh Clock Register

The CDRAM register (Figure 9) is used to define the rate at which refresh requests will be internally generated. The CDRAM register is used to maintain the start-

ing value of a down counter, which decrements each falling edge of CLKOUT. When the counter decrements to 1, a refresh request is generated and the counter is again loaded with the value contained in the CDRAM register. Initially, however, the contents of the CDRAM register is loaded into the down counter when the enable bit in the EDRAM register set. Thus, if the CDRAM register is changed, the new value will take effect when either the down counter reaches 1 and reloads itself, or whenever the E bit is written to a 1 (this is true whether the bit was previously set or not). When the 80C186 is reset, the CDRAM register is initialized to zero. A value of zero in the CDRAM register is used to indicate the maximum count rate of 512 clocks.

$$\frac{R_{\text{PERIOD}} (\mu\text{s}) \cdot \text{FREQ (MHz)}}{\# \text{Refresh Rows} + (\# \text{Refresh Rows} \cdot \% \text{Overhead})} = \text{CDRAM Register Value}$$

R_{Period} = Maximum Refresh period specified by the DRAM manufacturer (time in microseconds).
 FREQ = Operating Frequency at 80C186 in megahertz.
 $\# \text{Refresh Rows}$ = Total number of rows to be refreshed.
 $\% \text{Overhead}$ = Derating factor that estimates the number of missed refresh requests (typically 1–5%).

Figure 10. Equation to Calculate Refresh Interval

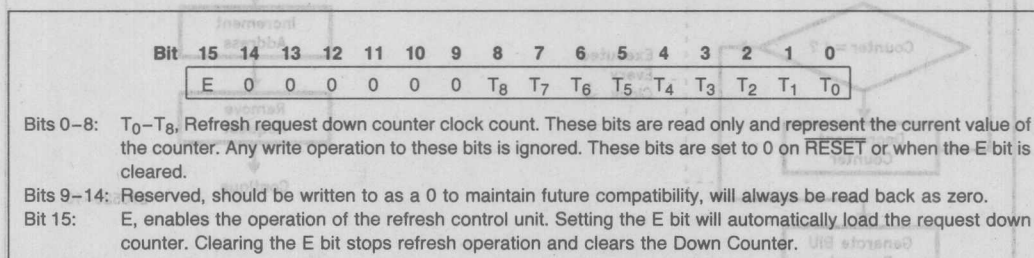


Figure 11. EDRAM Register Format

The equation shown in Figure 10 can be used to determine the value of the CDRAM register needed to establish a desired refresh request rate. Note that the equation is based on the internal operating frequency of the 80C186. Therefore, the request rate is effected by any change in operating frequency. Modification of the operating frequency can occur in two ways: modifying the input clock or entering power-save mode. There is no upper limitation as to the frequency of refresh requests (other than programming), but there is a lower limit. This lower limit is based on the fact that the request rate can be no faster than the time it takes to service the request. Subsequently, the minimum programming value of the CDRAM register should be 18 (12H). It is very doubtful that this will ever become a problem when operating at normal frequencies, since the refresh rate of most dynamic memories is well above this minimum programming value.

However, when making use of the power-save feature of the 80C186, it is possible to lower the operating frequency such that it will prevent adequate refreshing rates. When operating at 12.5 MHz, dividing the clock by 16 results in a cycle time of 1.28 microseconds. Since the minimum value of the CDRAM is 18, the minimum refresh rate is 23.04 microseconds. 23 microseconds is not fast enough to service most dynamic memories. Therefore, caution must be exercised when using the power-save feature of the 80C186. When there is a need to keep dynamic memory alive, the clock should not be divided much below 2 MHz to avoid monopolizing the bus with refresh activity. If there is no desire to keep memory alive during power-save operation, then the refresh unit can simply be disabled during this time.

Programming the Refresh Enable Register

The EDRAM register (Figure 11) is used to enable and disable the refresh control unit. Furthermore, reading the register returns the current value of the down counter.

Setting the E bit enables the RCU and loads the value of the CDRAM register into the down counter. Whenever the E bit is cleared, the refresh control unit is disabled and the down counter is cleared. Disabling the refresh control unit does not change the contents of the refresh address counter (i.e. it is not cleared or initialized to any specific value). Thus, when the refresh unit is again enabled, the address generated will continue from where it left off. Resetting the 80C186 automatically clears the E bit. There are no refresh bus cycles during a reset.

The current value of the down counter, as well as the present state of the E bit can be examined whenever the EDRAM register is read. Any unused bits will be returned as zero. Whenever the E bit is cleared, the T_0 through T_8 bits will be read as zero.

REFRESH CONTROL UNIT OPERATION

Figure 12 illustrates the two major operational functions of the refresh control unit that are responsible for initiating and controlling DRAM refresh bus cycles.

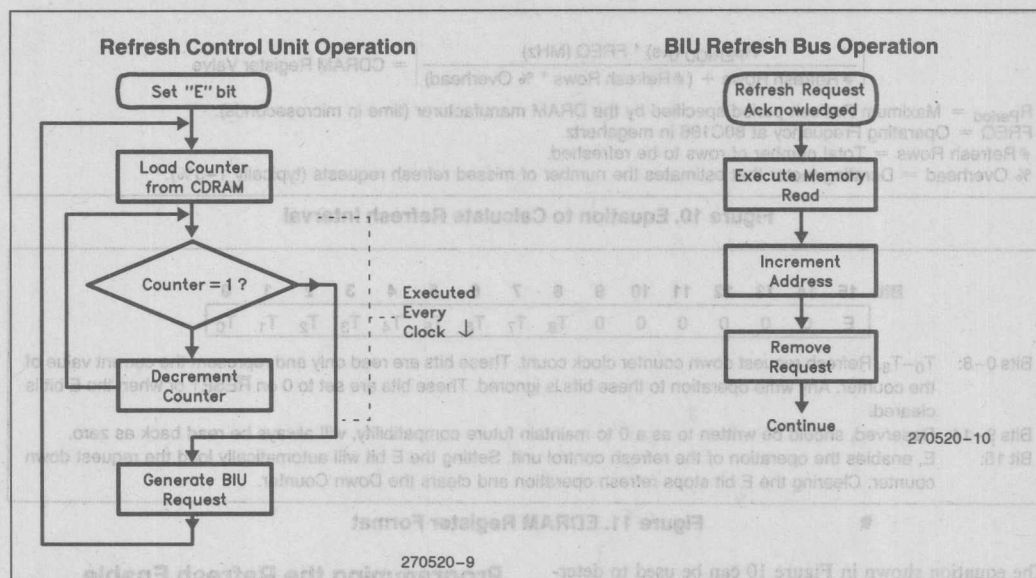


Figure 12. Flowchart of RCU Operation

The down counter is loaded (with the contents of the CDRAM register) on the falling edge of CLKOUT, either when the E bit is set or whenever the counter decrements to 1. Once loaded, the down counter will decrement every falling edge of CLKOUT. It will continue to decrement as long as the E bit remains set.

When the down counter finally decrements to 1, two things will happen. First, a request is generated to the BIU to run a refresh bus cycle. The request remains active until the bus cycle is run. Second, the down counter is reloaded with the value contained in the CDRAM register. At this time, the down counter will again begin counting down every clock cycle, it does not wait until the request has been serviced. This is done to ensure that each refresh request occurs at the correct interval. Otherwise, if the down counter only started after the previous request were service, the time between refresh requests would also be a function of bus activity, which for the most part is unpredictable. When the BIU services the refresh request, it will clear the request and increment the refresh address.

80C188 Address Considerations

The physical address that is generated during a refresh bus cycle is shown in Figure 7, and it applies to both the 80C186 and 80C188. For the 80C188, this means that the lower address bit A0 will not toggle during refresh operation. Since the 80C188 has an 8-bit external bus, A0 is used as part of memory address decoding. Whereas the 80C186, with its 16-bit external bus,

uses A0 (along with BHE) to select memory banks. Therefore, when designing 80C188 memory subsystems it is important not to include A0 as part of the ROW address that is used as a refresh address. Appendix A illustrates Memory Address Multiplexing Techniques that can be applied to the 80C186 and the 80C188.

MISSING REFRESH REQUESTS

Under most operating conditions, the frequency of refresh requests is a small percentage of the bus bandwidth. Still, there are several conditions that may prevent a refresh request from being serviced before another request is generated. These conditions include:

- 1) LOCKED Bus Cycles
- 2) Long Bus accesses (wait states)
- 3) Bus HOLD

LOCKED Bus Cycles

Whenever the bus is LOCKED, the CPU maintains control of the BIU and will not relinquish it until the locked operation is complete. Therefore, internal operations like refresh and DMA are not allowed to execute until the LOCKED instruction has completed. Where this presents the greatest problem is when an instruction such as a move string is executed, and is locked. The move string instruction can take from several clocks to hundreds of thousands of clocks to complete. Obviously anything that takes longer than 512 clocks to complete will always cause a refresh overflow.

Care should be taken not to generate long executing instructions that require bus accesses and are locked. The refresh request interval can be shortened to compensate for missing requests.

Long Bus Accesses

The 80C186 does not provide any mechanism to abort or terminate a bus access in the event ready is not returned within a specified amount of time (the 80C186 will infinitely wait for ready). Therefore, if a bus access is in progress when a refresh request is generated, the bus access must complete before the request will be serviced.

Bus HOLD

Special consideration is given when a refresh request is generated and the 80C186 is currently being held off the bus due to a HOLD request.

When another bus master has control of the bus, the HLDA signal is kept active as long as the HOLD input remains active. If a refresh request is generated while HOLD is active, the 80C186 will remove (drive inactive) the HLDA signal to indicate to the other bus master that the 80C186 wishes to regain control of the bus (see Figure 13). If, and only if, the HOLD input is removed will the BIU begin to run the refresh bus cycle.

Therefore, it is the responsibility of the system designer to ensure that the 80C186 can regain the bus if a refresh request is signaled. The sequence of HLDA going inactive while HOLD is active can be used to signal a pending refresh request. HOLD need only go inactive for

one clock period to allow the refresh bus cycle to be run. If HOLD is again asserted, the 80C186 will give up the bus after the refresh bus cycle has been run (provided there is not another refresh request generated during that time).

EFFECTS OF MISSING REFRESH REQUESTS

If a refresh request has not been serviced before another request is generated, the new request is not recorded and is lost. For instance, if the interval between refresh request is 15 microseconds and one request is lost, then the time between two requests will be 30 microseconds when the next request is finally serviced. In this example, missing one request will add 15 μ s to the total refresh time. If it is anticipated that refresh requests may be missed (due to programming or system operation), then the refresh request interval should be shortened to allow for missed requests.

Since the BIU is responsible for maintaining the refresh address counter, missing a refresh requests does not imply that refresh addresses are skipped. In fact, an address can never be skipped unless a reset occurs.

CONCLUSION

The Internal Refresh Control Unit of the 80C186 and 80C188 helps solve three issues concerning DRAM refreshing: a way to generate periodic refresh requests; a way to generate refresh addresses; a way to simplify DRAM memory controllers. Once a memory controller has been designed to handle the simple tasks of reading and writing the task of refreshing has already been built in.

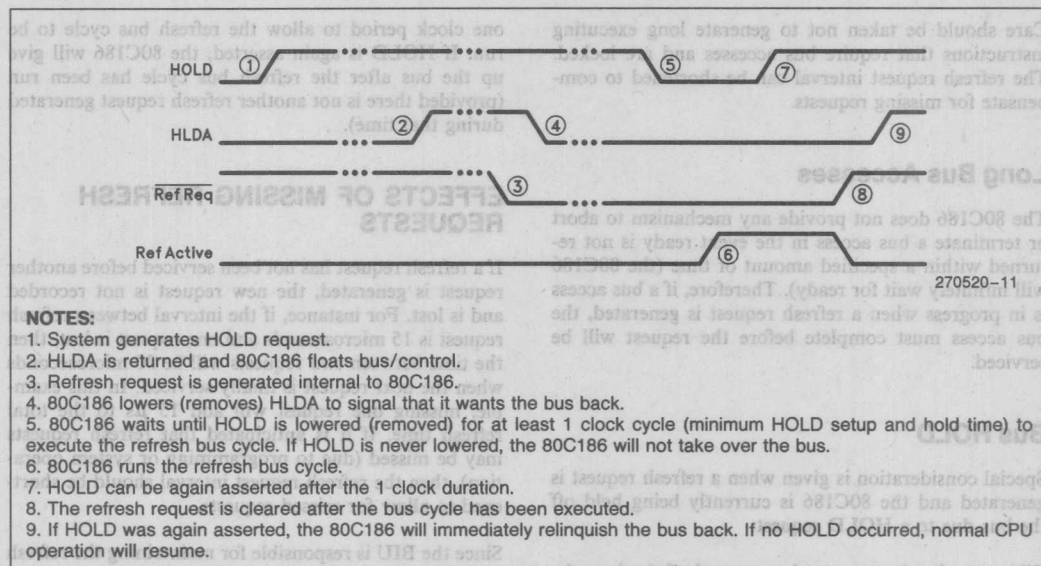


Figure 13. HOLD/HLDA Timing and Refresh Request

APPENDIX A

TYPICAL DRAM ADDRESS GENERATION CONSIDERATIONS FOR 80C186/80C188

80C186 DESIGNS

		Row Address (A0-AX)	Column Address (A0-AX)
64K x 1	(128K Bytes)	A1-A8	A9-A16
16K x 4	(32K Bytes)	A1-A8	A9-A14
256K x 1	(512K Bytes)	A1-A9	A10-A18
64K x 4	(128K Bytes)	A1-A8	A9-A16
1M x 1	(2M Bytes)	A1-A10	A11-A19 (+ Bank)
256K x 4	(512K Bytes)	A1-A9	A10-A18

80C188 DESIGNS

NOTE:

Address bit A0 can be used in either RAS or CAS addresses, so long as it is not included in any refresh address bits.

		Row Address (A0-AX)	Column Address (A0-AX)
64K x 1	(64K Bytes)	A1-A7, A0	A8-A15
16K x 4	(16K Bytes)	A1-A7, A0	A8-A13
256K x 1	(256K Bytes)	A1-A8, A0	A9-A17
64K x 4	(64K Bytes)	A1-A8	A0, A9-A15
1M x 1	(1M Byte)	A1-A9, A0	A10-A19
256K x 4	(256K Bytes)	A1-A9	A0, A10-A17

RAM Type	RAS Add	CAS Add	Refresh Add
64K x 1	A0-A7	A0-A7	A0-A6
16K x 4	A0-A7	A0-A5	A0-A6
256K x 1	A0-A8	A0-A8	A0-A7
64K x 4	A0-A7	A0-A7	A0-A7
1M x 1	A0-A9	A0-A9	A0-A8
256K x 4	A0-A8	A0-A8	A0-A8

APPENDIX A
TYPICAL DRAM ADDRESS GENERATION
CONSIDERATIONS FOR 80C186/80C188

80C186 DESIGNS

Row Address (A0-A7)	Column Address (A0-A7)		
A0-A7	A0-A7	(128K Bytes)	August 1990
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	

80C188 DESIGNS

Row Address (A0-A7)	Column Address (A0-A7)		
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	

DRAM Refresh/Control with the 80186/80188

Row Address (A0-A7)	Column Address (A0-A7)		
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	
A0-A7	A0-A7	(128K Bytes)	

STEVE FARRER
APPLICATIONS ENGINEER

DRAM REFRESH/ CONTROL WITH THE 80186/80188

BUS OVERHEAD

The absolute maximum overhead can be calculated at a given speed by taking the number of refresh cycles divided by the total number of bus cycles for a given period of time. At 8 MHz these values can be calculated as follows:

$$\frac{2 \text{ bus cycles}}{18.2 \text{ ns} / 800 \text{ ns}} \times 100 = 8.8\% \text{ maximum overhead}$$

In reality, the bus overhead associated with the DMA cycles is much lower due to the instruction prefetch queue. When a DMA cycle is requested by the timer for a refresh cycle, the bus interface unit honors the request on the next bus cycle boundary (with the exception of LOCKed bus cycles and odd aligned accesses). Typically, this time is idle time on the bus and the impact on the overall performance is extremely small. The following table shows more realistic data which was acquired by running 6 different benchmarks with and without the DMA channel enabled to provide refresh every 18.2µs.

BENCHMARK RESULTS @ 8 MHz

	Minimum	Maximum	Average
80186	1.3%	5.9%	2.5%
80188	2.4%	8.8%	3.4%

The program which showed the highest bus overhead tended to be very bus intensive. Also note that at faster frequencies the bus overhead becomes even less.

DMA OPERATION

The DMA controller is programmed to be source synchronized with the TC (transfer count) bit cleared. This ensures that the DMA controller never reaches a final count. The source pointer continues to increment through memory on every cycle. When RSTPFR is reached, the address rolls over to 0000H.

The programming values for the DMA registers are shown in Figure 1. The source pointer may be initialized to any location since the starting location of the refresh is arbitrary.

CONTENTS PAGE

THEORY OF OPERATION 5-74

READY LOGIC WITH MEMORY 5-74

BUS OVERHEAD 5-74

DMA OPERATION 5-74

TIMER OPERATION 5-76

EXAMPLE 1: DRAM CONTROL WITH A DELAY LINE 5-77

EXAMPLE 2: DRAM CONTROL WITH A PAL* 5-78

TIMING EQUATIONS 5-80

The control logic of the DRAM is such that a RAS (row address strobe) occurs on every memory refresh (row address of the address). This is necessary because the DMA channel is cycling through the entire 1 Mbyte address space and the address of the refresh cycle does not always fall within the range of the DRAM bank.

Although the address may be outside the DRAM range, the lower address bits continue to change and roll over to provide the row address.

READY LOGIC WITH MEMORY

Since the DMA controller is cycling through the entire 1 Mbyte address space, care must be taken to ensure that a READY signal is available for all addresses. One way to do this is to use only the internal wait state generator for memory areas and to strap the SRDY and ARDY pins HIGH. Whenever a refresh cycle occurs outside of a predefined internal wait state area, the external ready pins, which are active HIGH, will complete the bus cycle.

If it is necessary to use the external ready signals for certain memory regions, then it will be necessary to add logic which will generate a ready signal whenever the address of a refresh cycle falls where there is no memory. This can easily be accomplished by either decoding a couple of high order address lines, or by AND-ing

In many low-cost 80186/80188 designs, dynamic memory offers an excellent cost/performance advantage. However, DRAM interfacing is often complicated by the need to perform memory refreshing. This application brief describes how to use the Timer and DMA functionality of the 80186/80188 to perform memory refresh.

THEORY OF OPERATION

Dynamic RAM refreshing is accomplished by strobing a ROW address to every ROW of the DRAM within a given period of time. One way to do this is to perform periodic sequential reads to the DRAM using a DMA controller and a Timer. This can be achieved with the 80186/188 by Programming Timer 2 and one of the DMA channels such that the timer generated one DMA cycle approximately every 15 micro-seconds. Please note that this is a single row refresh method and not a burst refresh. Single row refreshing reduces the bus overhead considerably when compared to burst refreshing.

The control logic of the DRAM is such that a RAS (row address strobe) occurs on every memory read, regardless of the address. This is necessary because the DMA channel is cycling through the entire 1 MByte address space and the address of the refresh cycle does not always fall within the range of the DRAM bank.

Although the address may be outside the DRAM range, the lower address bits continue to change and roll over to provide the row address.

READY LOGIC WITH MEMORY

Since the DMA controller is cycling through the entire 1 MByte address space, care must be taken to ensure that a READY signal is available for all addresses. One way to do this is to use only the internal wait state generator for memory areas and to strap the SRDY and ARDY pins HIGH. Whenever a refresh cycle occurs outside of a predefined internal wait state area, the external ready pins, which are active HIGH, will complete the bus cycle.

If it is necessary to use the external ready signals for certain memory regions, then it will be necessary to add logic which will generate a ready signal whenever the address of a refresh cycle falls where there is no memory. This can easily be accomplished by either decoding a couple of high order address lines, or by AND-ing

all the chip selects so that READY goes active whenever all the memory chip selects are inactive (i.e. the cycle is not in a valid memory region).

BUS OVERHEAD

The absolute maximum overhead can be calculated at a given speed by taking the number of refresh cycles divided by the total number of bus cycles for a given period of time. At 8 MHz these values can be calculated as follows:

$$\frac{2 \text{ bus cycles}}{15.2 \mu\text{s}/500 \text{ ns}} \times 100 = 6.6\% \text{ maximum overhead}$$

In reality, the bus overhead associated with the DMA cycles is much lower due to the instruction prefetch queue. When a DMA cycle is requested by the timer for a refresh cycle, the Bus Interface Unit honors the request on the next bus cycle boundary (with the exception of LOCKed bus cycles and odd aligned accesses). Typically this time is idle time on the bus and the impact on the overall performance is extremely small. The following table shows more realistic data which was acquired by running 6 different benchmarks with and without the DMA channel enabled to provide refresh every 15.2μs.

BENCHMARK RESULTS @ 8 MHz

	Minimum	Maximum	Average
80186	1.3%	5.9%	2.5%
80188	2.4%	6.5%	3.4%

The programs which showed the highest bus overhead tended to be very bus intensive. Also note that at faster frequencies the bus overhead becomes even less.

DMA OPERATION

The DMA controller is programmed to be source synchronized with the TC (transfer count) bit cleared. This ensures that the DMA controller never reaches a final count. The source pointer continues to increment through memory on every cycle. When FFFFFH is reached, the address rolls over to 00000H.

The programming values for the DMA registers are shown in Figure 1. The source pointer may be initialized to any location since the starting location of the refresh is arbitrary.

The value of the Transfer Count register is also arbitrary since the TC bit is not set. The DMA channel will continue to run cycles upon request from Timer 2 even after the Transfer Count register has reached zero. Once zero is reached, the Transfer Count register will roll over to FFFFH and continue to count down.

The destination pointer may be set to any available memory or I/O location. This pointer must be set so that it neither increments nor decrements. Otherwise, the address of the deposit cycle would cycle through memory or I/O doing writes which could possibly be destructive. Thus the INC and DEC bits of the control register should be cleared.

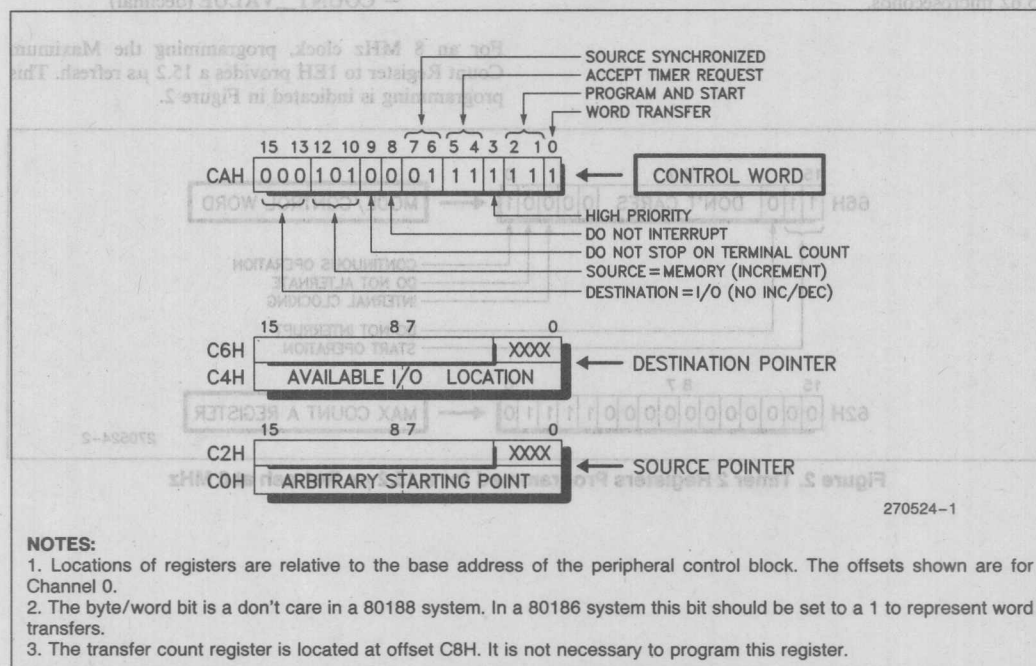


Figure 1. DMA Registers

TIMER OPERATION

Timer 2 must be programmed to generate a DMA request every time a row must be refreshed. Since we are not using a burst refresh, the refresh time is divided up evenly among the number of rows. For a 2 ms refresh DRAM with 128 rows, the time between rows equals 15.62 microseconds.

When setting the count value of the timer, keep in mind the timer clock is operating at one-fourth the CPU clock frequency. Thus, the equation for setting the timer count is:

$$\frac{(\text{CPU CLOUT FREQ}) \times (\text{Time Between ROWS})}{4} = \text{COUNT_VALUE (decimal)}$$

For an 8 MHz clock, programming the Maximum Count Register to 1EH provides a 15.2 μ s refresh. This programming is indicated in Figure 2.

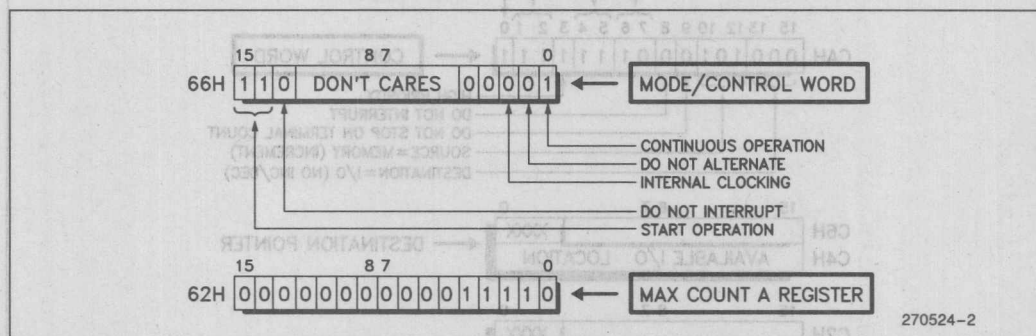


Figure 2. Timer 2 Registers Programmed for a 15.2 μ s Refresh at 8 MHz

EXAMPLE 1: DRAM CONTROL WITH A DELAY LINE

This is the most straight forward way of implementing the $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ logic. A $\overline{\text{RAS}}$ signal is generated by either $\overline{\text{RD}}$ or $\overline{\text{WR}}$ going active while the address is within the corresponding range. Normally the logic for $\overline{\text{RAS}}$ would also go active for a refresh cycle status, but since this information is not available on the 80186/80188, a $\overline{\text{RAS}}$ must be generated for every $\overline{\text{RD}}$ and $\overline{\text{WR}}$, regardless address.

The $\overline{\text{MUX}}$ signal is used to change from the $\overline{\text{RAS}}$ address to the $\overline{\text{CAS}}$ address after latching with $\overline{\text{RAS}}$. This is accomplished by using a delay line which generates a $\overline{\text{MUX}}$ signal by a fixed number of nano-seconds after $\overline{\text{RAS}}$ is generated. The important timing here is the necessary hold time for the row address into the DRAM.

The $\overline{\text{MUX}}$ signal is initially HIGH which sends the A side (see Figure 3) Row address through the multiplex-

er to the DRAM. This address consists of A0 through A7. The B address (A8 through A16) is selected when $\overline{\text{MUX}}$ goes LOW. The system shown in Figure 3 represents that of an 80188 system.

For an 80186 system, the A address would start at A1. The least significant address line A0 along with $\overline{\text{BHE}}$ would be used to decode $\overline{\text{WE}}$ into $\overline{\text{WEH}}$ and $\overline{\text{WEL}}$ which will be shown in the second example. Also, the 186 DMA must be set to do word transfers so that the address is incremented by 2 after each refresh cycle. This is necessary to ensure A1 increments by 1 every refresh cycle.

$\overline{\text{CAS}}$ is generated in the same manner by delaying the $\overline{\text{MUX}}$ signal a fixed number of nano-seconds. Typically $\overline{\text{CAS}}$ goes inactive at the same time as $\overline{\text{RAS}}$ to ensure a valid $\overline{\text{CAS}}$ precharge time before the next DRAM access. The 80186/188 chip selects are used to ensure that $\overline{\text{CAS}}$ only goes active when the address falls within the DRAM bank range, and to ensure that $\overline{\text{CAS}}$ does not go active during I/O cycles.

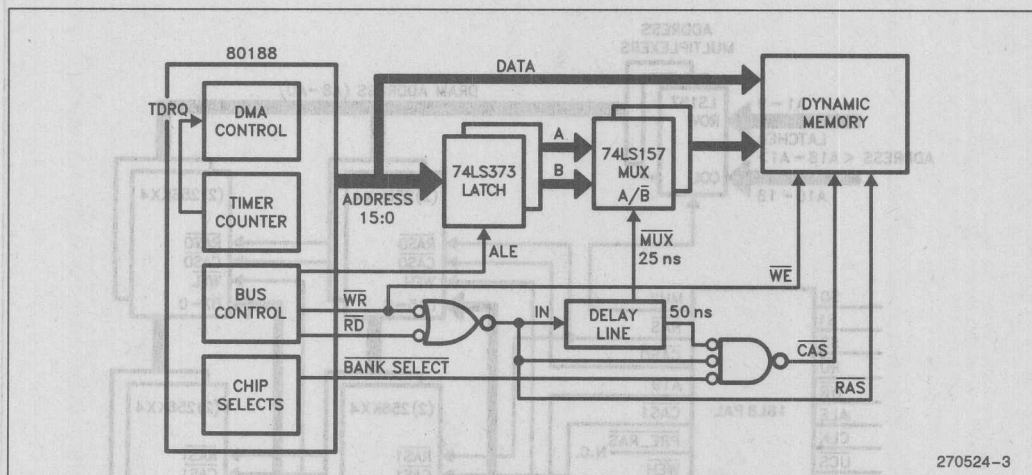


Figure 3. Using A Delay Line for DRAM Control

This design uses a PAL to generate all the control logic for the DRAM array. Internal feedback is used on the signals to control the timing and states of the $\overline{\text{RAS}}$, $\overline{\text{MUX}}$ and $\overline{\text{CAS}}$ signals.

This design uses 256k X 4 DRAMs. With minor changes to the PAL equations this design could just as easily make use of 64k X 1, 64k X 4, or 256k X 1 DRAMs.

The $\overline{\text{RAS}}$ signal is generated off ALE going LOW, bus cycle status active, and $\overline{\text{PRE_RAS}}$ being active. The $\overline{\text{PRE_RAS}}$ signal is necessary to ensure that a RAS is not accidentally generated when S2-S0 are becoming valid and ALE has not yet gone HIGH in T4 phase 2. $\overline{\text{PRE_RAS}}$ does not go active until ALE has gone HIGH.

$\overline{\text{RAS}}$ is initiated for every memory read and write regardless of the bus cycle address. This ensures a row

refresh when the refresh address falls outside of the DRAM bank and also a refresh to both banks simultaneously so that the frequency of the refresh can be set for the number of rows in one bank of DRAM.

The $\overline{\text{UCS}}$ (Upper Chip Select) from the 80186/188 is used to disable DRAM signals when the processor is attempting to access upper memory control ROM. Thus the portion of memory used by the $\overline{\text{UCS}}$ (maximum 256k) is unavailable in the upper DRAM. However, the RAS signal must still be allowed during $\overline{\text{UCS}}$ access to ensure refreshing when the DMA refresh cycle occurs in the $\overline{\text{UCS}}$ region.

$\overline{\text{MUX}}$ is generated off T2 phase 1 and $\overline{\text{RAS}}$ active. $\overline{\text{MUX}}$ will remain low until the current $\overline{\text{RAS}}$ signal goes inactive during T3 phase 2.

$\overline{\text{CAS0}}$ and $\overline{\text{CAS1}}$ are generated off $\overline{\text{MUX}}$ being active and T2 phase 2 of the bus cycle. $\overline{\text{CAS}}$ goes inactive at the start of T4 phase 2.

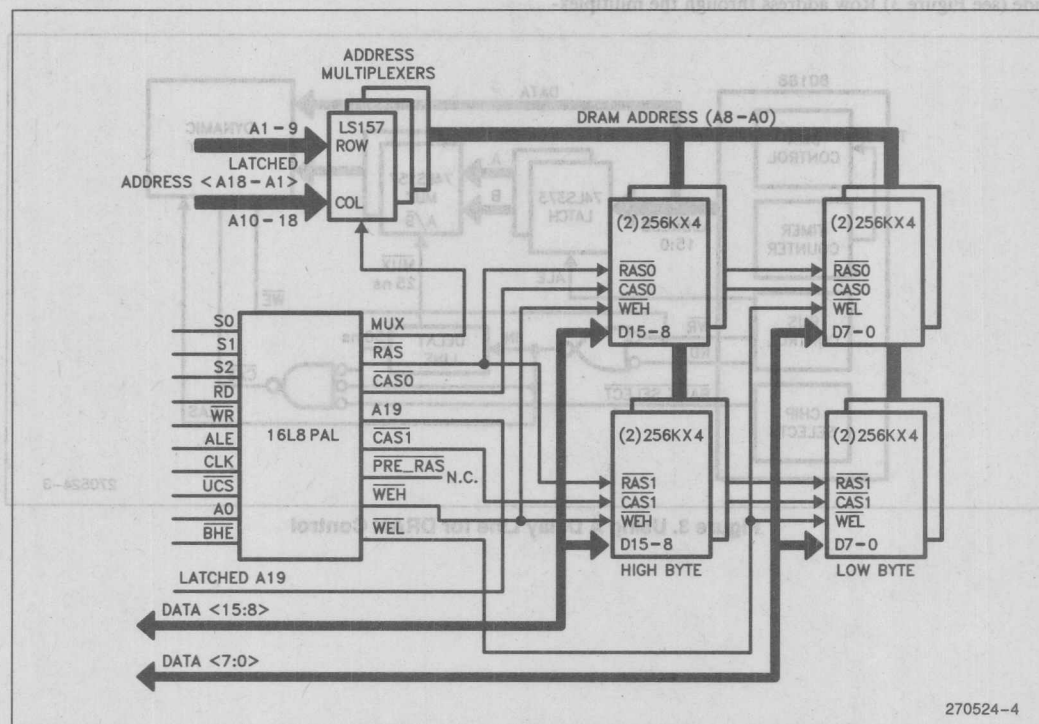


Figure 4. Using a PAL for DRAM Control

*PAL® is a registered trademark of Monolithic Memories.

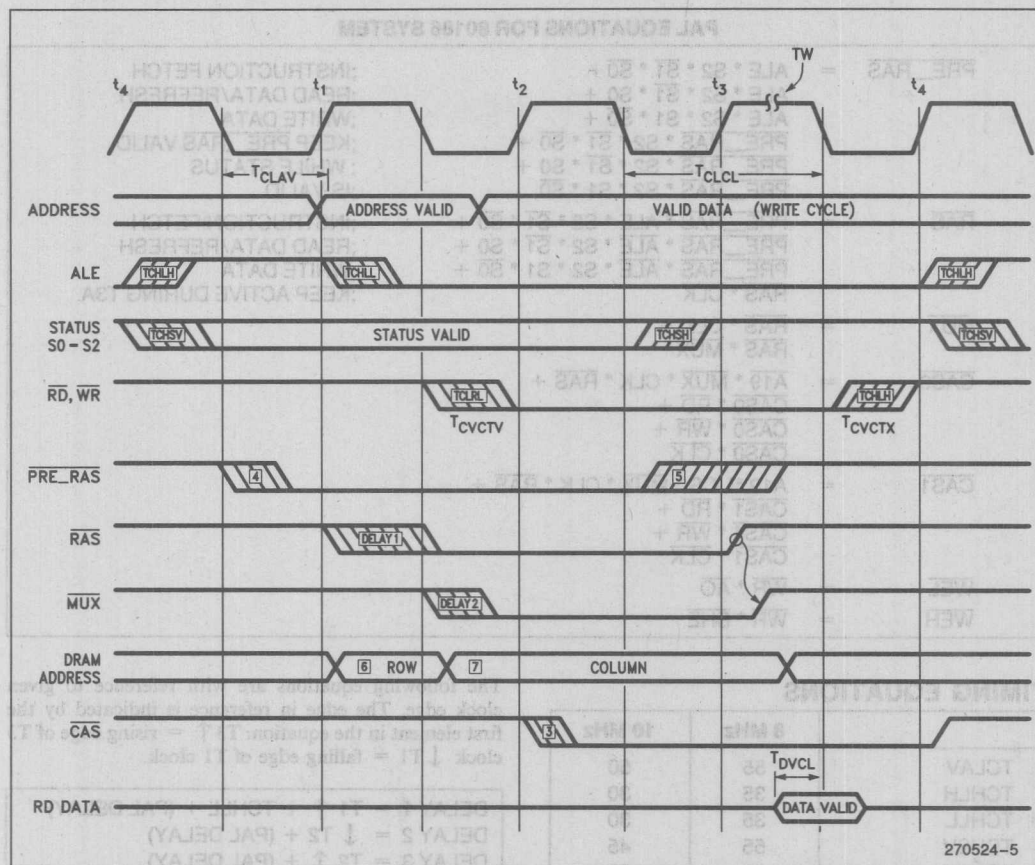


Figure 5. Timing Diagram for PAL DRAM Controller

5

PAL EQUATIONS FOR 80186 SYSTEM

PRE_RAS	=	ALE * S2 * $\overline{S1}$ * $\overline{S0}$ + ALE * S2 * $\overline{S1}$ * S0 + ALE * S2 * S1 * $\overline{S0}$ + PRE_RAS * S2 * $\overline{S1}$ * $\overline{S0}$ + PRE_RAS * S2 * $\overline{S1}$ * S0 + PRE_RAS * S2 * S1 * $\overline{S0}$;INSTRUCTION FETCH ;READ DATA/REFRESH ;WRITE DATA ;KEEP PRE_RAS VALID ; WHILE STATUS ;IS VALID
RAS	=	PRE_RAS * ALE * S2 * $\overline{S1}$ * $\overline{S0}$ + PRE_RAS * ALE * S2 * $\overline{S1}$ * S0 + PRE_RAS * ALE * S2 * S1 * $\overline{S0}$ + RAS * CLK	;INSTRUCTION FETCH ;READ DATA/REFRESH ;WRITE DATA ;KEEP ACTIVE DURING T3A
MUX	=	RAS * CLK + RAS * MUX	
CAS0	=	A19 * MUX * CLK * \overline{RAS} + CAS0 * \overline{RD} + CAS0 * \overline{WR} + CAS0 * CLK	
CAS1	=	A19 * UCS * MUX * CLK * \overline{RAS} + CAS1 * \overline{RD} + CAS1 * \overline{WR} + CAS1 * CLK	
WEL	=	WR * AO	
WEH	=	WR * BHE	

TIMING EQUATIONS

	8 MHz	10 MHz
TCLAV	55	50
TCHLH	35	30
TCHLL	35	30
TCHSV	55	45
TCLSH	65	50
TCLRL/TCVCTV	70	56
TCLRH	55	44
TDVCL	20	15

The following equations are with reference to given clock edge. The edge in reference is indicated by the first element in the equation: T3 \uparrow = rising edge of T3 clock \downarrow T1 = falling edge of T1 clock.

DELAY 1 = T1 \uparrow + TCHLL + (PAL DELAY)
 DELAY 2 = \downarrow T2 + (PAL DELAY)
 DELAY 3 = T2 \uparrow + (PAL DELAY)
 DELAY 4 = \downarrow T1 + (PAL DELAY)
 DELAY 5 = \downarrow T3 + TCLSH + (PAL DELAY)
 DELAY 6 = \downarrow T1 + TCLAV + (MUX DELAY)
 DELAY 7 = \downarrow T2 + DELAY 2 + (MUX DELAY)

ACCESS TIME FROM \overline{RAS} = 2.5 (TCLCL) - DELAY 1 - TDVCL

ACCESS TIME FROM \overline{CAS} = 1.5 (TCLCL) - DELAY 3 - TDVCL

MCS[®]-96 Application Notes & 6

Article Reprint



APPLICATION NOTE

AP-248

September 1987

Using The 8096

IRA HORDEN

MCO APPLICATIONS ENGINEER

6

Order Number: 270061-002

Using The 8096 CONTENTS PAGE

1.0 INTRODUCTION 6-5

2.0 8096 OVERVIEW 6-5

2.1. General Description 6-5

2.1.1. CPU Section 6-6

2.1.2. I/O Features 6-8

2.2. The Processor Section 6-8

2.2.1. Operations and Addressing
Modes 6-8

2.2.2. Assembly Language 6-11

2.2.3. Interrupts 6-12

2.3. On-Chip I/O Section 6-14

2.3.1. Timer/Counters 6-14

2.3.2. HSI 6-15

2.3.3. HSO 6-16

2.3.4. Serial Port 6-17

2.3.5. A to D Converter 6-20

2.3.6. PWM Register 6-21

3.0 BASIC SOFTWARE EXAMPLES 6-23

3.1. Using the 8096's Processing
Section 6-23

3.1.1. Table Interpolation 6-23

3.1.2. PL/M-96 6-26

3.2. Using the I/O Section 6-28

3.2.1. Using the HSI Unit 6-28

3.2.2. Using the HSO Unit 6-29

3.2.3. Using the Serial Port in
Mode 1 6-33

3.2.4. Using the A to D 6-35

4.0 ADVANCED SOFTWARE
EXAMPLES 6-35

4.1. Simultaneous I/O Routines under
Interrupt Control 6-35

4.2. Software Serial Port Using the
HSIO Unit 6-38

4.3. Interfacing an Optical Encoder to
the HSI Unit 6-43

5.0 HARDWARE EXAMPLE 6-55

5.1. EPROM Only Minimum
System 6-55

5.2. Port Reconstruction 6-57

6.0 CONCLUSION 6-58

7.0 BIBLIOGRAPHY 6-58

CONTENTS

PAGE

APPENDICES

Appendix A. Basic Software Examples

A.1. Table Lookup 1	6-59
A.2. Table Lookup 2	6-61
A.3. PLM-96 Code with Expansion	6-63
A.4. Pulse Measurement	6-69
A.5. Enhanced Pulse Measurement	6-71
Enhanced HSI Pulse Measurement	6-71
Generating a PWM with the HSO	6-73
Changes to Declarations for HSO	6-73
Driver Module for HSO PWM	6-73
Using the Serial Port in Mode 1	6-73
Scanning the A to D Channels	6-73
Using Multiple I/O Devices	6-73
Software Serial Port	6-73
Declarations	6-73
Software Serial Port Interface	6-73
Software Serial Port Initialization	6-73
Software Serial Port Transmit	6-73
Process	6-73
Receive Process	6-73
Motor Control HSO 0 Timer	6-73
Motor Control HSI Data Available	6-73
Motor Control Mode 1 Routines	6-73
Motor Control Mode 0 Routines	6-73
Motor Control Software Timer 1	6-73
Motor Control Next Position	6-73
Lookup	6-73
Motor Control Timer Interrupt	6-73
Motor Control Software Timer	6-73
Interrupt Handler	6-73
Motor Control Software Timer 2	6-73

CONTENTS

PAGE

A.6. PWM Using the HSO	6-73
A.7. Serial Port	6-77
A.8. A to D Converter	6-79
Appendix B. HSO and A to D Under Interrupt Control	6-81
Appendix C. Software Serial Port	6-85
Appendix D. Motor Control Program	6-91

Figures

2-1.	8096 Block Diagram	6-5
2-2.	Memory Map	6-6
2-3.	SFR Layout	6-7
2-4.	Major I/O Functions	6-8
2-5.	Instruction Summary	6-9
2-6.	Instruction Format	6-11
2-7.	Interrupt Sources	6-12
2-8.	Interrupt Vectors and Priorities	6-12
2-9.	Interrupt Structure Block Diagram	6-13
2-10.	The PSW Register	6-14
2-11.	HSI Unit Block Diagram	6-15
2-12.	HSI Mode Register	6-15
2-13.	HSO Command Register	6-16
2-14.	HSO Block Diagram	6-16
2-15.	Serial Port Control/Status Register	6-17
2-16.	Baud Rate Formulas	6-18
2-17.	Baud Rate Values for 10, 11, 12 MHz	6-19
2-18.	Multiprocessor Communication	6-20
2-19.	A to D Result/Command Register	6-21
2-20.	PWM Output Waveforms	6-22
2-21.	PWM to Analog Conversion Circuitry	6-22
3-1.	Using the HSIO to Monitor Rotating Machinery	6-32
3-2.	Serial Port Level Conversion	6-34
4-1.	10-Bit Asynchronous Frame	6-39
4-2.	Optical Encoder and Waveforms	6-43
4-3.	Filtered Encoder Waveforms	6-44
4-4.	Schematic of Optical Encoder to 8096 Interface	6-45
4-5.	Motor Driver Circuitry	6-45
4-6.	Mode State Diagram	6-48
4-7.	Motor Control Modes	6-53
5-1.	Minimum System Configuration	6-56

Listings

3-1.	Include File DEMO96.INC	6-23
3-2.	ASM-96 Code for Table Lookup Routine 1	6-24
3-3.	ASM-96 Code for Table Lookup Routine 1	6-25
3-4.	PLM-96 Code for Table Lookup Routine 1	6-27
3-5.	32-Bit Result Multiply Procedure for PLM-96	6-27
3-6.	Measuring Pulses Using the HSI Unit	6-28
3-7.	Enhanced HSI Pulse Measurement Routine	6-29
3-8.	Generating a PWM with the HSO ..	6-30
3-9.	Changes to Declarations for HSO Routine	6-31
3-10.	Driver Module for HSO PWM Program	6-31
3-11.	Using the Serial Port in Mode 1	6-33
3-12.	Scanning the A to D Channels	6-35
4-1.	Using Multiple I/O Devices	6-36
4-2.	Software Serial Port Declarations	6-39
4-3.	Software Serial Port Interface Routines	6-40
4-4.	Software Serial Port Initialization Routine	6-40
4-5.	Software Serial Port Transmit Process	6-41
4-6.	Receive Process	6-41
4-7.	Motor Control HSO.0 Timer Routine	6-46
4-8.	Motor Control HSI Data Available Routine	6-48
4-9.	Motor Control Mode 1 Routines	6-49
4-10.	Motor Control Mode 0 Routines	6-50
4-11.	Motor Control Software Timer 1 Routine	6-51
4-12.	Motor Control Next Position Lookup	6-53
4-13.	Motor Control Timer Interrupt Routine	6-54
4-14.	Motor Control Software Timer Interrupt Handler	6-54
4-15.	Motor Control Software Timer 2 Routine	6-55

1.0 INTRODUCTION

High speed digital signals are frequently encountered in modern control applications. In addition, there is often a requirement for high speed 16-bit and 32-bit precision in calculations. The MCS®-96 product line, generically referred to as the 8096, is designed to be used in applications which require high speed calculations and fast I/O operations.

The 8096 is a 16-bit microcontroller with dedicated I/O subsystems and a complete set of 16-bit arithmetic instructions including multiply and divide operations. This Ap-note will briefly describe the 8096 in section 2, and then give short examples of how to use each of its key features in section 3. The concluding sections feature a few examples which make use of several chip features simultaneously and some hardware connection suggestions. Further information on the 8096 and its use is available from the sources listed in the bibliography.

2.0 8096 OVERVIEW

2.1. General Description

Unlike microprocessors, microcontrollers are generally optimized for specific applications. Intel's 8048 was optimized for general control tasks while the 8051 was optimized for 8-bit math and single bit boolean operations. The 8096 has been designed for high speed/high performance control applications. Because it has been designed for these applications the 8096 architecture is different from that of the 8048 or 8051.

There are two major sections of the 8096; the CPU section and the I/O section. Each of these sections can be subdivided into functional blocks as shown in Figure 2-1.

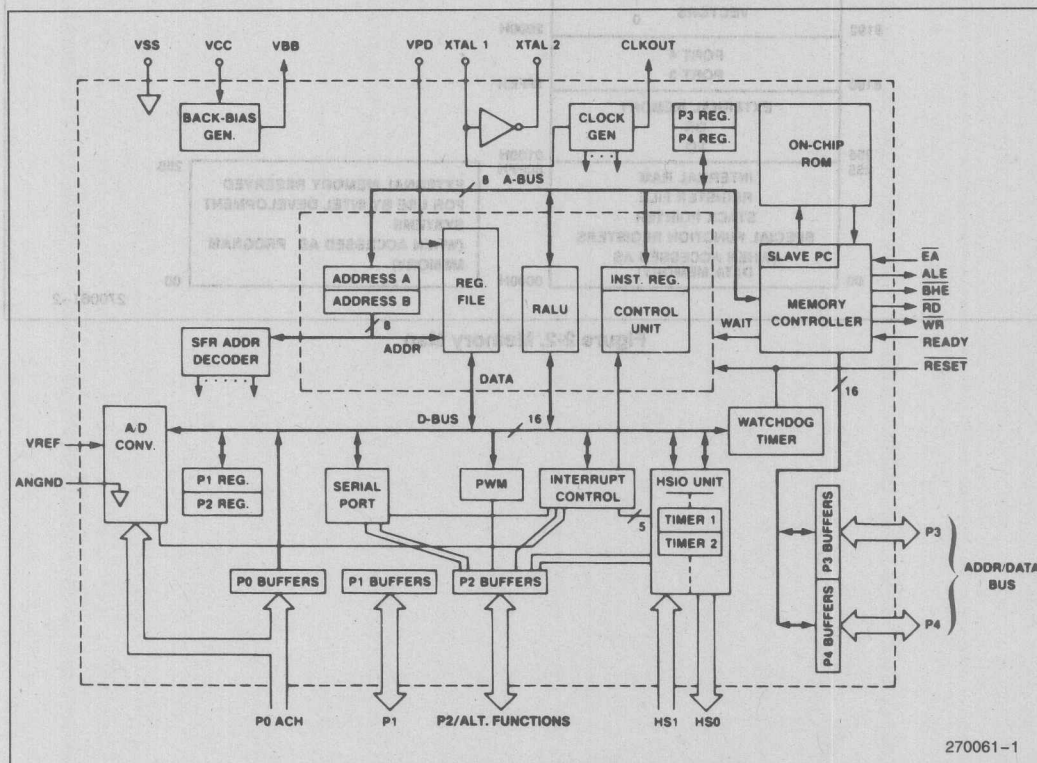


Figure 2-1. 8096 Block Diagram

2.1.1. CPU SECTION

The CPU of the 8096 uses a 16-bit ALU which operates on a 256-byte register file instead of an accumulator. Any of the locations in the register file can be used for sources or destinations for most of the instructions. This is called a register to register architecture. Many of the instructions can also use bytes or words from anywhere in the 64K byte address space as operands. A memory map is shown in Figure 2-2.

In the lower 24 bytes of the register file are the register-mapped I/O control locations, also called Special Function Registers or SFRs. These registers are used to control the on-chip I/O features. The remaining 232 bytes are general purpose RAM, the upper 16 of which can be kept alive using a low current power-down mode.

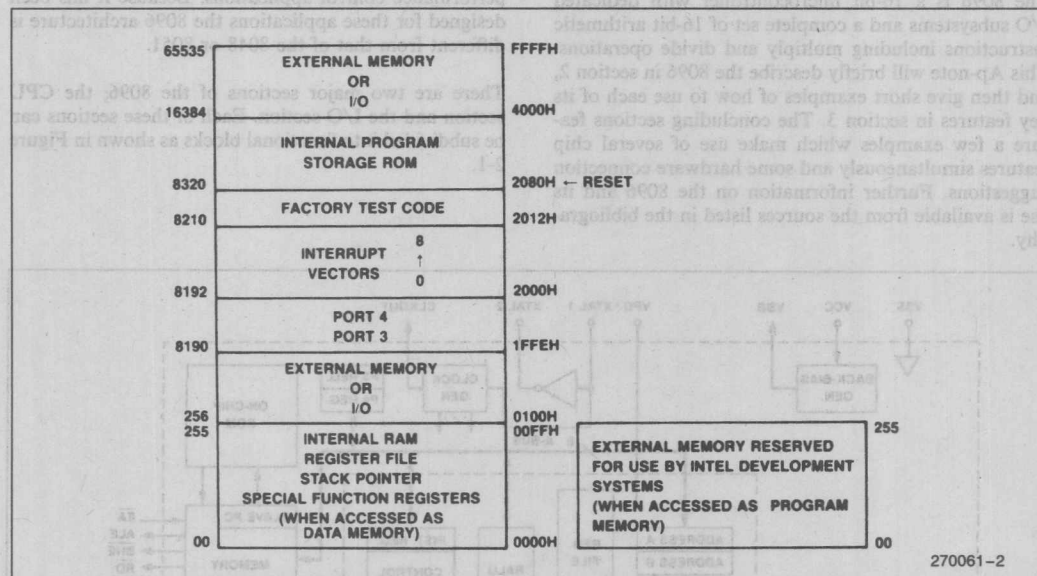


Figure 2-2. Memory Map

Figure 2-3 shows the layout of the register mapped I/O. Some of these registers serve two functions, one if they are read from and another if they are written

to. More information about the use of these registers is included in the description of the features which they control.

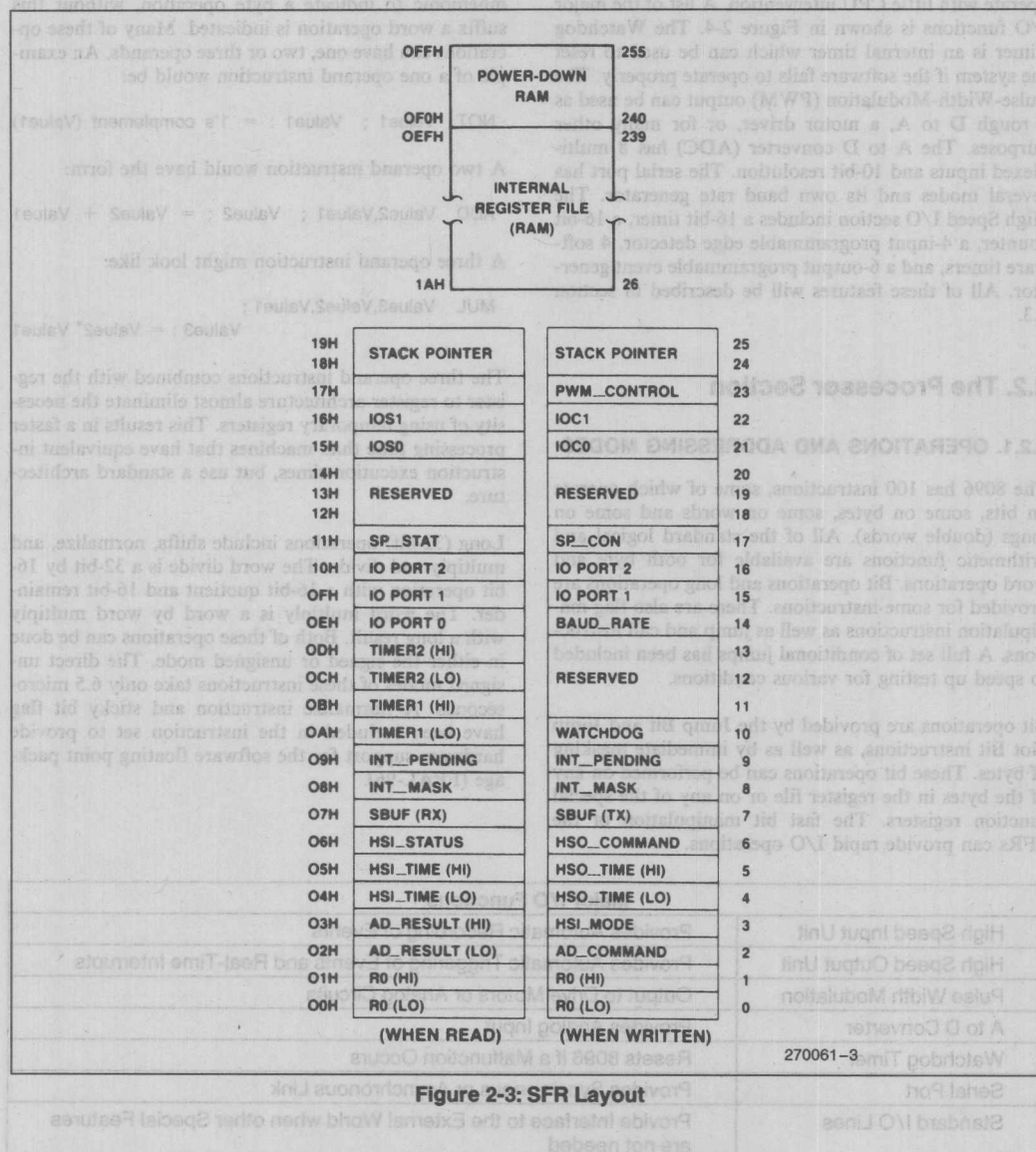


Figure 2-3: SFR Layout

2.1.2. I/O FEATURES

Many of the I/O features on the 8096 are designed to operate with little CPU intervention. A list of the major I/O functions is shown in Figure 2-4. The Watchdog Timer is an internal timer which can be used to reset the system if the software fails to operate properly. The Pulse-Width-Modulation (PWM) output can be used as a rough D to A, a motor driver, or for many other purposes. The A to D converter (ADC) has 8 multiplexed inputs and 10-bit resolution. The serial port has several modes and its own baud rate generator. The High Speed I/O section includes a 16-bit timer, a 16-bit counter, a 4-input programmable edge detector, 4 software timers, and a 6-output programmable event generator. All of these features will be described in section 2.3.

2.2. The Processor Section

2.2.1. OPERATIONS AND ADDRESSING MODES

The 8096 has 100 instructions, some of which operate on bits, some on bytes, some on words and some on longs (double words). All of the standard logical and arithmetic functions are available for both byte and word operations. Bit operations and long operations are provided for some instructions. There are also flag manipulation instructions as well as jump and call instructions. A full set of conditional jumps has been included to speed up testing for various conditions.

Bit operations are provided by the Jump Bit and Jump Not Bit instructions, as well as by immediate masking of bytes. These bit operations can be performed on any of the bytes in the register file or on any of the special function registers. The fast bit manipulation of the SFRs can provide rapid I/O operations.

A symmetric set of byte and word operations make up the majority of the 8096 instruction set. The assembly language for the 8096 (ASM-96) uses a "B" suffix on a mnemonic to indicate a byte operation, without this suffix a word operation is indicated. Many of these operations can have one, two or three operands. An example of a one operand instruction would be:

NOT Value1 ; Value1 : = 1's complement (Value1)

A two operand instruction would have the form:

ADD Value2,Value1 ; Value2 : = Value2 + Value1

A three operand instruction might look like:

MUL Value3,Value2,Value1 ;
Value3 : = Value2 * Value1

The three operand instructions combined with the register to register architecture almost eliminate the necessity of using temporary registers. This results in a faster processing time than machines that have equivalent instruction execution times, but use a standard architecture.

Long (32-bit) operations include shifts, normalize, and multiply and divide. The word divide is a 32-bit by 16-bit operation with a 16-bit quotient and 16-bit remainder. The word multiply is a word by word multiply with a long result. Both of these operations can be done in either the signed or unsigned mode. The direct unsigned modes of these instructions take only 6.5 microseconds. A normalize instruction and sticky bit flag have been included in the instruction set to provide hardware support for the software floating point package (FPAL-96).

Major I/O Functions	
High Speed Input Unit	Provides Automatic Recording of Events
High Speed Output Unit	Provides Automatic Triggering of Events and Real-Time Interrupts
Pulse Width Modulation	Output to Drive Motors or Analog Circuits
A to D Converter	Provides Analog Input
Watchdog Timer	Resets 8096 if a Malfunction Occurs
Serial Port	Provides Synchronous or Asynchronous Link
Standard I/O Lines	Provide Interface to the External World when other Special Features are not needed

Figure 2-4. Major I/O Functions

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	$D \leftarrow D + A$	✓	✓	✓	✓	↑	—	
ADD/ADDB	3	$D \leftarrow B + A$	✓	✓	✓	✓	↑	—	
ADDC/ADDCB	2	$D \leftarrow D + A + C$	↓	✓	✓	✓	↑	—	
SUB/SUBB	2	$D \leftarrow D - A$	✓	✓	✓	✓	↑	—	
SUB/SUBB	3	$D \leftarrow B - A$	✓	✓	✓	✓	↑	—	
SUBC/SUBCB	2	$D \leftarrow D - A + C - 1$	↓	✓	✓	✓	↑	—	
CMP/CMPB	2	$D - A$	✓	✓	✓	✓	↑	—	
MUL/MULU	2	$D, D + 2 \leftarrow D * A$	—	—	—	—	—	?	2
MUL/MULU	3	$D, D + 2 \leftarrow B * A$	—	—	—	—	—	?	2
MULB/MULUB	2	$D, D + 1 \leftarrow D * A$	—	—	—	—	—	?	3
MULB/MULUB	3	$D, D + 1 \leftarrow B * A$	—	—	—	—	—	?	3
DIVU	2	$D \leftarrow (D, D + 2)/A, D + 2 \leftarrow \text{remainder}$	—	—	—	✓	↑	—	2
DIVUB	2	$D \leftarrow (D, D + 1)/A, D + 1 \leftarrow \text{remainder}$	—	—	—	✓	↑	—	3
DIV	2	$D \leftarrow (D, D + 2)/A, D + 2 \leftarrow \text{remainder}$	—	—	—	?	↑	—	2
DIVB	2	$D \leftarrow (D, D + 1)/A, D + 1 \leftarrow \text{remainder}$	—	—	—	?	↑	—	3
AND/ANDB	2	$D \leftarrow D \text{ and } A$	✓	✓	0	0	—	—	
AND/ANDB	3	$D \leftarrow B \text{ and } A$	✓	✓	0	0	—	—	
OR/ORB	2	$D \leftarrow D \text{ or } A$	✓	✓	0	0	—	—	
XOR/XORB	2	$D \leftarrow D \text{ (excl. or) } A$	✓	✓	0	0	—	—	
LD/LDB	2	$D \leftarrow A$	—	—	—	—	—	—	
ST/STB	2	$A \leftarrow D$	—	—	—	—	—	—	
LDBSE	2	$D \leftarrow A; D + 1 \leftarrow \text{SIGN}(A)$	—	—	—	—	—	—	3, 4
LDBZE	2	$D \leftarrow A; D + 1 \leftarrow 0$	—	—	—	—	—	—	3, 4
PUSH	1	$SP \leftarrow SP - 2; (SP) \leftarrow A$	—	—	—	—	—	—	
POP	1	$A \leftarrow (SP); SP \leftarrow SP + 2$	—	—	—	—	—	—	
PUSHF	0	$SP \leftarrow SP - 2; (SP) \leftarrow PSW;$ $PSW \leftarrow 0000H$	0	0	0	0	0	0	
POPF	0	$PSW \leftarrow (SP); SP \leftarrow SP + 2; I \leftarrow 0$	✓	✓	✓	✓	✓	✓	
SJMP	1	$PC \leftarrow PC + 11\text{-bit offset}$	—	—	—	—	—	—	5
LJMP	1	$PC \leftarrow PC + 16\text{-bit offset}$	—	—	—	—	—	—	5
BR (indirect)	1	$PC \leftarrow (A)$	—	—	—	—	—	—	
SCALL	1	$SP \leftarrow SP - 2; (SP) \leftarrow PC;$ $PC \leftarrow PC + 11\text{-bit offset}$	—	—	—	—	—	—	5
LCALL	1	$SP \leftarrow SP - 2; (SP) \leftarrow PC;$ $PC \leftarrow PC + 16\text{-bit offset}$	—	—	—	—	—	—	5
RET	0	$PC \leftarrow (SP); SP \leftarrow SP + 2$	—	—	—	—	—	—	
J (conditional)	1	$PC \leftarrow PC + 8\text{-bit offset (if taken)}$	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

Figure 2-5. Instruction Summary

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Mnemonic	Oper- ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D – 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D – 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 – D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	✓	✓	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb ———— lsb ← 0	✓	?	✓	✓	↑	—	7
SHR/SHRB/SHRL	2	0 → msb ———— lsb → C	✓	?	✓	0	—	✓	7
SHRA/SHRAB/SHRAL	2	msb → msb ———— lsb → C	✓	✓	✓	0	—	✓	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left Shift Till msb = 1; D ← shift count	✓	?	0	—	—	—	7
TRAP	0	SP ← SP – 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

Figure 2-5. Instruction Summary (Continued)

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

One operand of most of the instructions can be used with any one of six addressing modes. These modes increase the flexibility and overall execution speed of the 8096. The addressing modes are: register-direct, immediate, indirect, indirect with auto-increment, and long and short indexed.

The fastest instruction execution is gained by using either register direct or immediate addressing. Register-direct addressing is similar to normal direct addressing, except that only addresses in the register file or SFRs can be addressed. The indexed mode is used to directly address the remainder of the 64K address space. Immediate addressing operates as would be expected, using the data following the opcode as the operand.

Both of the indirect addressing modes use the value in a word register as the address of the operand. If the indirect auto-increment mode is used then the word register is incremented by one after a byte access or by two after a word access. This mode is particularly useful for accessing lookup tables.

Access to any of the locations in the 64K address space can be obtained by using the long indexed addressing

mode. In this mode a 16-bit 2's complement value is added to the contents of a word register to form the address of the operand. By using the zero register as the index, ASM96 (the assembler) can accept "direct" addressing to any location. The zero register is located at 0000H and always has a value of zero. A short indexed mode is also available to save some time and code. This mode uses an 8-bit 2's complement number as the offset instead of a 16-bit number.

2.2.2. ASSEMBLY LANGUAGE

The multiple addressing modes of the 8096 make it easy to program in assembly language and provide an excellent interface to high level languages. The instructions accepted by the assembler consist of mnemonics followed by either addresses or data. A list of the mnemonics and their functions are shown in Figure 2-5. The addresses or data are given in different formats depending on the addressing mode. These modes and formats are shown in Figure 2-6.

Additional information on 8096 assembly language is available in the MCS-96 Macro Assembler Users Guide, listed in the bibliography.

Mnem	Dest or Src1	: One operand direct
Mnem	Dest, Src1	: Two operand direct
Mnem	Dest, Src1, Src2	: Three operand direct
Mnem	#Src1	: One operand immediate
Mnem	Dest, #Src1	: Two operand immediate
Mnem	Dest, Src1, #Src2	: Three operand immediate
Mnem	[addr]	: One operand indirect
Mnem	[addr] +	: One operand indirect auto-increment
Mnem	Dest, [addr]	: Two operand indirect
Mnem	Dest, [addr] +	: Two operand indirect auto-increment
Mnem	Dest, Src1, [addr]	: Three operand indirect
Mnem	Dest, Src1, [addr] +	: Three operand indirect auto-increment
Mnem	Dest, offs [addr]	: Two operand indexed (short or long)
Mnem	Dest, Src1, offs [addr]	: Three operand indexed (short or long)

Where: "Mnem" is the instruction mnemonic
 "Dest" is the destination register
 "Src1", "Src2" are the source registers
 "addr" is a register containing a value to be used in computing the address of an operand
 "offs" is an offset used in computing the address of an operand

Figure 2-6. Instruction Format

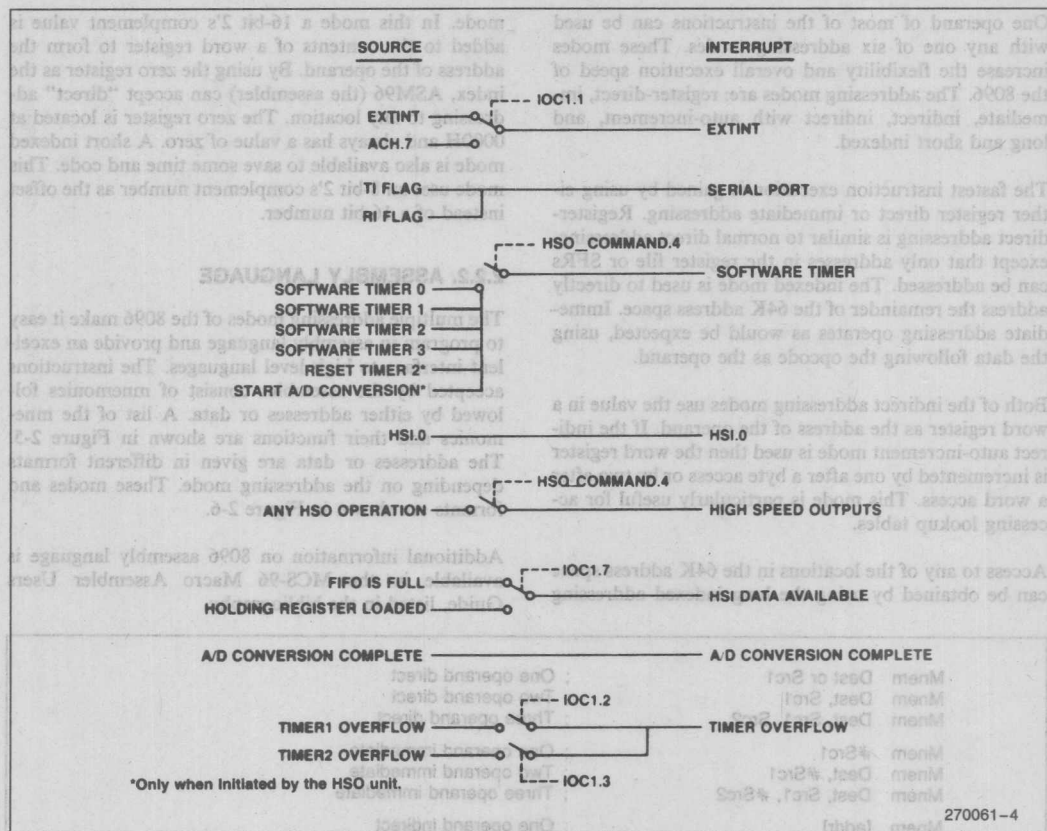


Figure 2-7. Interrupt Sources

2.2.3. INTERRUPTS

The flexibility of the instruction set is carried through into the interrupt system. There are 20 different interrupt sources that can be used on the 8096. The 20 sources vector through 8 locations or interrupt vectors. The vector names and their sources are shown in Figure 2-7, with their locations listed in Figure 2-8. Control of the interrupts is handled through the Interrupt Pending Register (INT_PENDING), the Interrupt Mask Register (INT_MASK), and the I bit in the PSW (PSW.9). Figure 2-9 shows a block diagram of the interrupt structure. The INT_PENDING register contains bits which get set by hardware when an interrupt occurs. If the interrupt mask register bit for that source is a 1 and PSW.9 = 1, a vector will be taken to the address listed in the interrupt vector table for that

Source	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 2-8. Interrupt Vectors and Priorities

source. When the vector is taken the INT_PENDING bit is cleared. If more than one bit is set in the INT_PENDING register with the corresponding bit set in the INT_MASK register, the Interrupt with the highest priority shown in Figure 2-8 will be executed.

The software can make the hardware interrupts work in almost any fashion desired by having each routine run with its own setup in the INT_MASK register. This will be clearly seen in the examples in section 4 which change the priority of the vectors in software. The

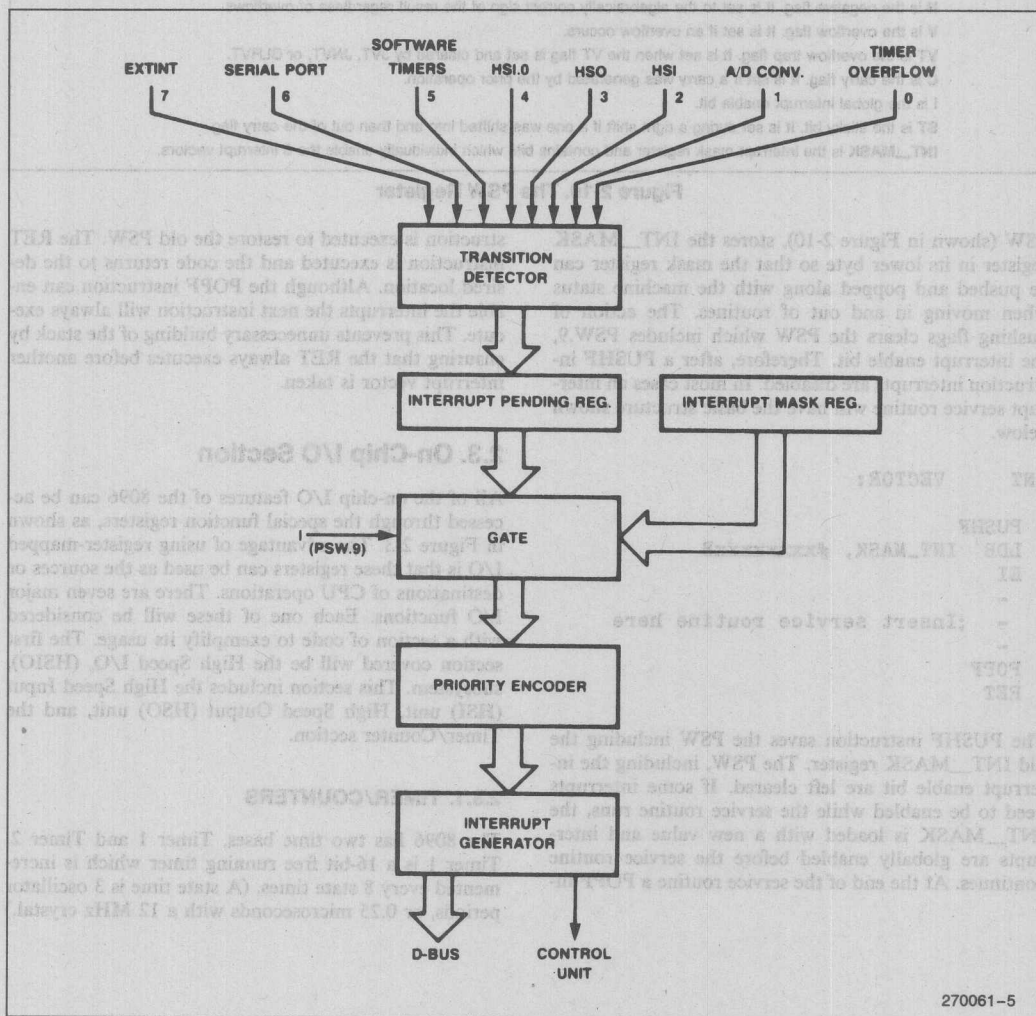


Figure 2-9. Interrupt Structure Block Diagram

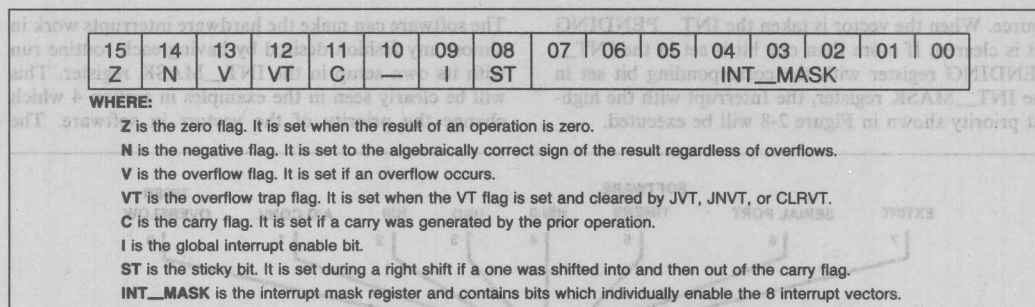


Figure 2-10. The PSW Register

PSW (shown in Figure 2-10), stores the INT_MASK register in its lower byte so that the mask register can be pushed and popped along with the machine status when moving in and out of routines. The action of pushing flags clears the PSW which includes PSW.9, the interrupt enable bit. Therefore, after a PUSHF instruction interrupts are disabled. In most cases an interrupt service routine will have the basic structure shown below.

```

INT    VECTOR:

PUSHF
LDB   INT_MASK, #xxxxxxx
EI
-
-    ;Insert service routine here
-
POPF
RET

```

The PUSHF instruction saves the PSW including the old INT_MASK register. The PSW, including the interrupt enable bit are left cleared. If some interrupts need to be enabled while the service routine runs, the INT_MASK is loaded with a new value and interrupts are globally enabled before the service routine continues. At the end of the service routine a POPF in-

struction is executed to restore the old PSW. The RET instruction is executed and the code returns to the desired location. Although the POPF instruction can enable the interrupts the next instruction will always execute. This prevents unnecessary building of the stack by ensuring that the RET always executes before another interrupt vector is taken.

2.3. On-Chip I/O Section

All of the on-chip I/O features of the 8096 can be accessed through the special function registers, as shown in Figure 2-3. The advantage of using register-mapped I/O is that these registers can be used as the sources or destinations of CPU operations. There are seven major I/O functions. Each one of these will be considered with a section of code to exemplify its usage. The first section covered will be the High Speed I/O, (HSIO), subsystem. This section includes the High Speed Input (HSI) unit, High Speed Output (HSO) unit, and the Timer/Counter section.

2.3.1. TIMER/COUNTERS

The 8096 has two time bases, Timer 1 and Timer 2. Timer 1 is a 16-bit free running timer which is incremented every 8 state times. (A state time is 3 oscillator periods, or 0.25 microseconds with a 12 MHz crystal.)

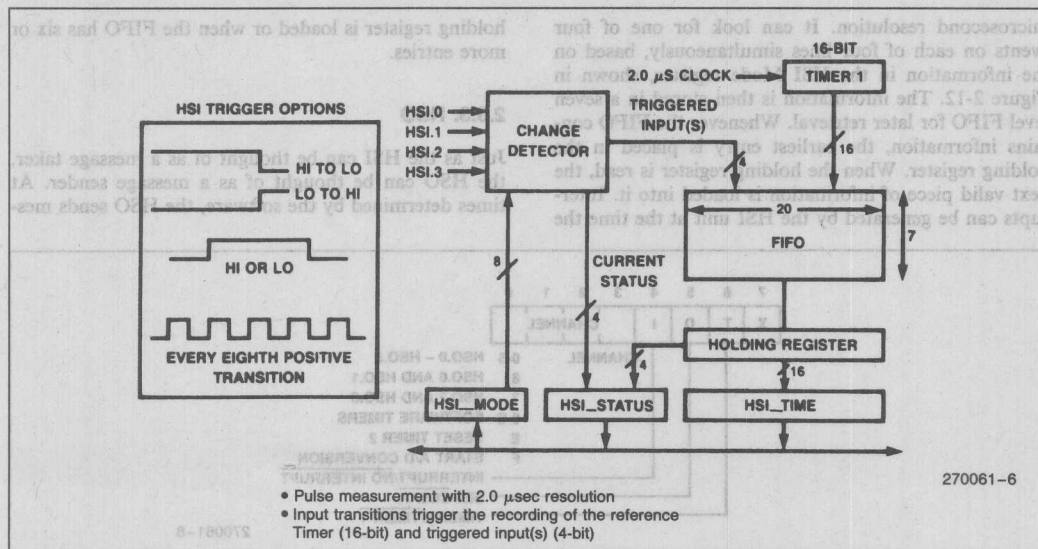


Figure 2-11. HSI Unit Block Diagram

Its value can be read at any time and used as a reference for both the HSI section and the HSO section. Timer 1 can cause an interrupt when it overflows, and cannot be modified or stopped without resetting the entire chip. Timer 2 is really an event counter since it uses an external clock source. Like Timer 1, it is 16-bits wide, can be read at any time, can be used with the HSO section, and can generate an interrupt when it overflows. Control of Timer 2 is limited to incrementing it and resetting it. Specific values can not be written to it.

Although the 8096 has only two timers, the timer flexibility is equal to a unit with many timers thanks to the HSIO unit. The HSI enables one to measure times of external events on up to four lines using Timer 1 as a timer base. The HSO unit can schedule and execute internal events and up to six external events based on the values in either Timer 1 or Timer 2. The 8096 also includes separate, dedicated timers for the baud rate generator and watchdog timer.

2.3.2. HSI

The HSI unit can be thought of as a message taker which records the line which had an event and the time at which the event occurred. Four types of events can trigger the HSI unit, as shown in the HSI block diagram in Figure 2-11. The HSI unit can measure pulse widths and record times of events with a 2

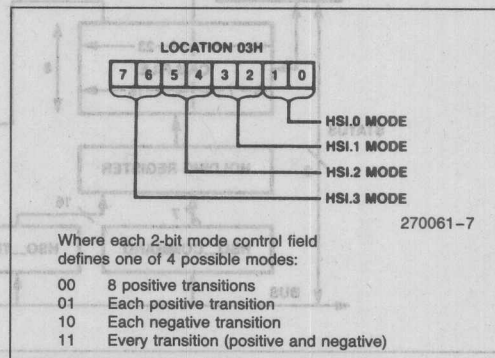


Figure 2-12. HSI Mode Register

microsecond resolution. It can look for one of four events on each of four lines simultaneously, based on the information in the HSI Mode register, shown in Figure 2-12. The information is then stored in a seven level FIFO for later retrieval. Whenever the FIFO contains information, the earliest entry is placed in the holding register. When the holding register is read, the next valid piece of information is loaded into it. Interrupts can be generated by the HSI unit at the time the

holding register is loaded or when the FIFO has six or more entries.

2.3.3. HSO

Just as the HSI can be thought of as a message taker, the HSO can be thought of as a message sender. At times determined by the software, the HSO sends mes-

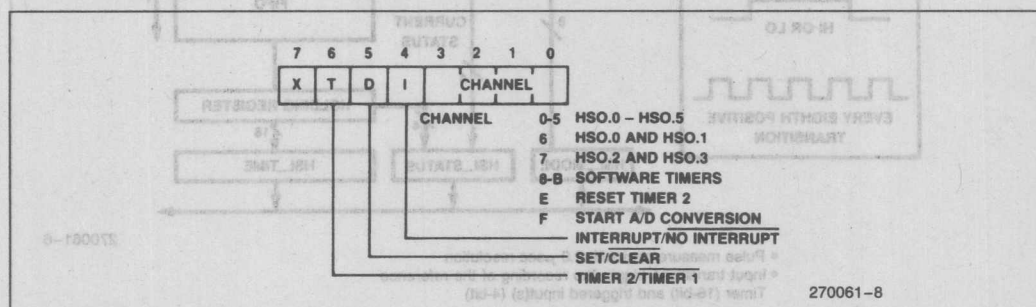


Figure 2-13. HSO Command Register

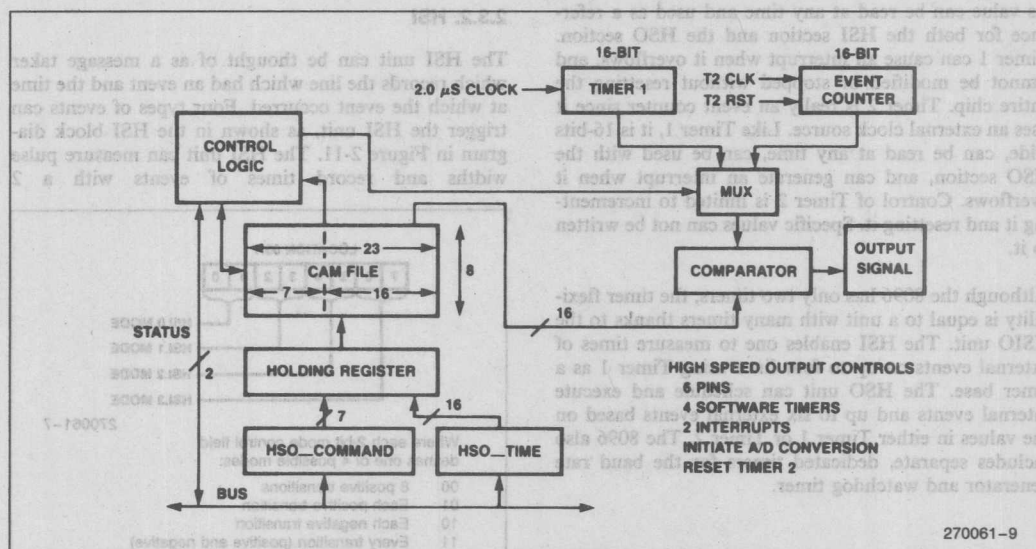


Figure 2-14. HSO Block Diagram

sages to various devices to have them turn on, turn off, start processing, or reset. Since the programmed times can be referenced to either Timer 1 or Timer 2, the HSO makes the two timers look like many. For example, if several events have to occur at specific times, the HSO unit can schedule all of the events based on a single timer. The events that can be scheduled to occur and the format of the command written to the HSO Command register are shown in Figure 2-13.

The software timers listed in the figure are actually 4 software flags in I/O Status Register 1 (IOS1). These flags can be set, and optionally cause an interrupt, at any time based on Timer 1 or Timer 2. In most cases these timers are used to trigger interrupt routines which must occur at regular intervals. A multitask process can easily be set up using the software timers.

A CAM (Content Addressable Memory) file is the main component of the HSO. This file stores up to eight events which are pending to occur. Every state time one location of the CAM is compared with the two timers. After 8 state times, (two microseconds with a 12 MHz clock), the entire CAM has been searched for time matches. If a match occurs the specified event will be triggered and that location of the CAM will be made available for another pending event. A block diagram of the HSO unit is shown in Figure 2-14.

2.3.4. Serial Port

Controlling a device from a remote location is a simple task that frequently requires additional hardware with many processors. The 8096 has an on-chip serial port to reduce the total number of chips required in the system.

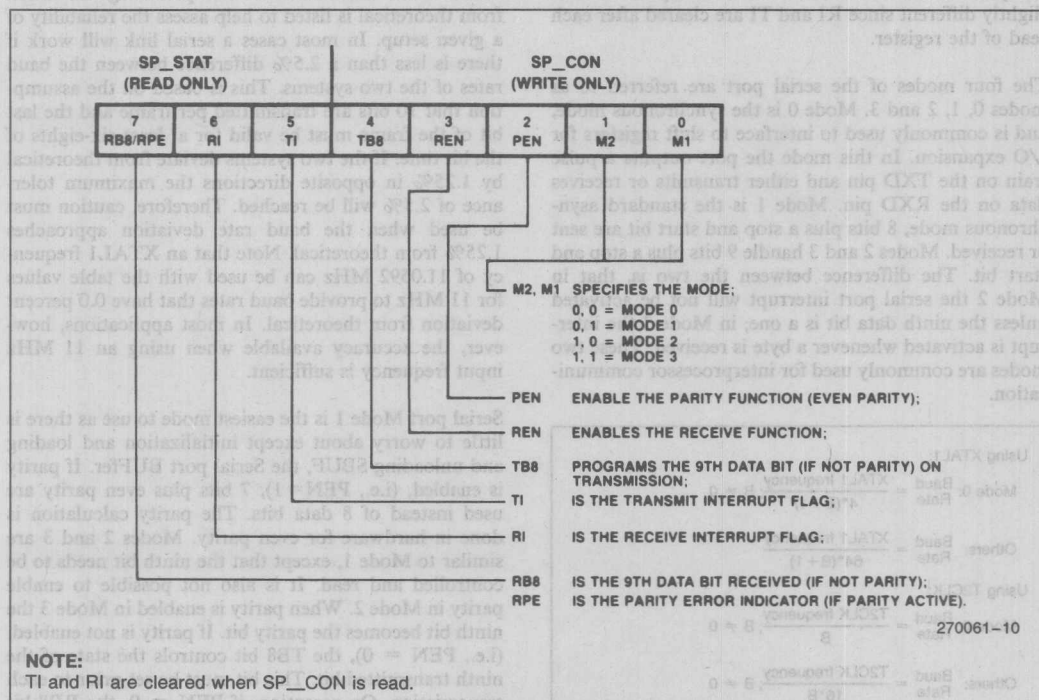


Figure 2-15. Serial Port Control/Status Register

The serial port is similar to that on the MCS-51 product line. It has one synchronous and three asynchronous modes. In the asynchronous modes baud rates of up to 187.5 Kbaud can be used, while in the synchronous mode rates up to 1.5 Mbaud are available. The chip has a baud rate generator which is independent of Timer 1 and Timer 2, so using the serial port does not take away any of the HSI, HSO or timer flexibility or functionality.

Control of the serial port is provided through the SPCON/SPSTAT (Serial Port CONTROL/Serial Port STATUS) register. This register, shown in Figure 2-15, has some bits which are read only and others which are write only. Although the functionality of the port is similar to that of the 8051, the names of some of the modes and control bits are different. The way in which the port is used from a software standpoint is also slightly different since RI and TI are cleared after each read of the register.

The four modes of the serial port are referred to as modes 0, 1, 2 and 3. Mode 0 is the synchronous mode, and is commonly used to interface to shift registers for I/O expansion. In this mode the port outputs a pulse train on the TXD pin and either transmits or receives data on the RXD pin. Mode 1 is the standard asynchronous mode, 8 bits plus a stop and start bit are sent or received. Modes 2 and 3 handle 9 bits plus a stop and start bit. The difference between the two is, that in Mode 2 the serial port interrupt will not be activated unless the ninth data bit is a one; in Mode 3 the interrupt is activated whenever a byte is received. These two modes are commonly used for interprocessor communication.

Using XTAL1:	
Mode 0: Baud Rate	$= \frac{\text{XTAL1 frequency}}{4 \cdot (B + 1)}; B \neq 0$
Others: Baud Rate	$= \frac{\text{XTAL1 frequency}}{64 \cdot (B + 1)}$
Using T2CLK:	
Mode 0: Baud Rate	$= \frac{\text{T2CLK frequency}}{B}; B \neq 0$
Others: Baud Rate	$= \frac{\text{T2CLK frequency}}{16 \cdot B}; B \neq 0$
Note that B cannot equal 0, except when using XTAL1 in other than mode 0.	

Figure 2-16. Baud Rate Formulas

Baud rates for all of the modes are controlled through the Baud Rate register. This is a byte wide register which is loaded sequentially with two bytes, and internally stores the value as a word. The least significant byte is loaded to the register followed by the most significant. The most significant bit of the baud value determines the clock source for the baud rate generator. If the bit is a one, the XTAL1 pin is used as the source, if it is a zero, the T2 CLK pin is used. The formulas shown in Figure 2-16 can be used to calculate the baud rates. The variable "B" is used to represent the least significant 15 bits of the value loaded into the baud rate register.

The baud rate register values for common baud rates are shown in Figure 2-17. These values can be used when XTAL1 is selected as the clock source for serial modes other than Mode 0. The percentage deviation from theoretical is listed to help assess the reliability of a given setup. In most cases a serial link will work if there is less than a 2.5% difference between the baud rates of the two systems. This is based on the assumption that 10 bits are transmitted per frame and the last bit of the frame must be valid for at least six-eighths of the bit time. If the two systems deviate from theoretical by 1.25% in opposite directions the maximum tolerance of 2.5% will be reached. Therefore, caution must be used when the baud rate deviation approaches 1.25% from theoretical. Note that an XTAL1 frequency of 11.0592 MHz can be used with the table values for 11 MHz to provide baud rates that have 0.0 percent deviation from theoretical. In most applications, however, the accuracy available when using an 11 MHz input frequency is sufficient.

Serial port Mode 1 is the easiest mode to use as there is little to worry about except initialization and loading and unloading SBUF, the Serial port BUFFER. If parity is enabled, (i.e., PEN = 1), 7 bits plus even parity are used instead of 8 data bits. The parity calculation is done in hardware for even parity. Modes 2 and 3 are similar to Mode 1, except that the ninth bit needs to be controlled and read. It is also not possible to enable parity in Mode 2. When parity is enabled in Mode 3 the ninth bit becomes the parity bit. If parity is not enabled, (i.e., PEN = 0), the TB8 bit controls the state of the ninth transmitted bit. This bit must be set prior to each transmission. On reception, if PEN = 0, the RB8 bit indicates the state of the ninth received bit. If parity is enabled, (i.e., PEN = 1), the same bit is called RPE (Receive Parity Error), and is used to indicate a parity error.

XTAL1 Frequency = 12.0 MHz		
Baud Rate	Baud Register Value	Percent Error
19.2K	8009H	+ 2.40
9600	8013H	+ 2.40
4800	8026H	- 0.16
2400	804DH	- 0.16
1200	809BH	- 0.16
300	8270H	0.00
XTAL1 Frequency = 11.0 MHz		
19.2K	8008H	+ 0.54
9600	8011H	+ 0.54
4800	8023H	+ 0.54
2400	8047H	+ 0.54
1200	808EH	- 0.16
300	823CH	+ 0.01
XTAL1 Frequency = 10.0 MHz		
19.2K	8007H	- 1.70
9600	800FH	- 1.70
4800	8020H	+ 1.38
2400	8040H	- 0.16
1200	8081H	- 0.16
300	8208H	+ 0.03

Figure 2-17. Baud Rate Values for 10, 11, 12 MHz

The software used to communicate between processors is simplified by making use of Modes 2 and 3. In a basic protocol the ninth bit is called the address bit. If it is set high then the information in that byte is either the address of one of the processors on the link, or a command for all the processors. If the bit is a zero, the byte contains information for the processor or processors previously addressed. In standby mode all processors wait in Mode 2 for a byte with the address bit set. When they receive that byte, the software determines if the next message is for them. The processor that is to

receive the message switches to Mode 3 and receives the information. Since this information is sent with the ninth bit set to zero, none of the processors set to Mode 2 will be interrupted. By using this scheme the overall CPU time required for the serial port is minimized.

A typical connection diagram for the multi-processor mode is shown in Figure 2-18. This type of communication can be used to connect peripherals to a desk top computer, the axis of a multi-axis machine, or any other group of microcontrollers jointly performing a task.

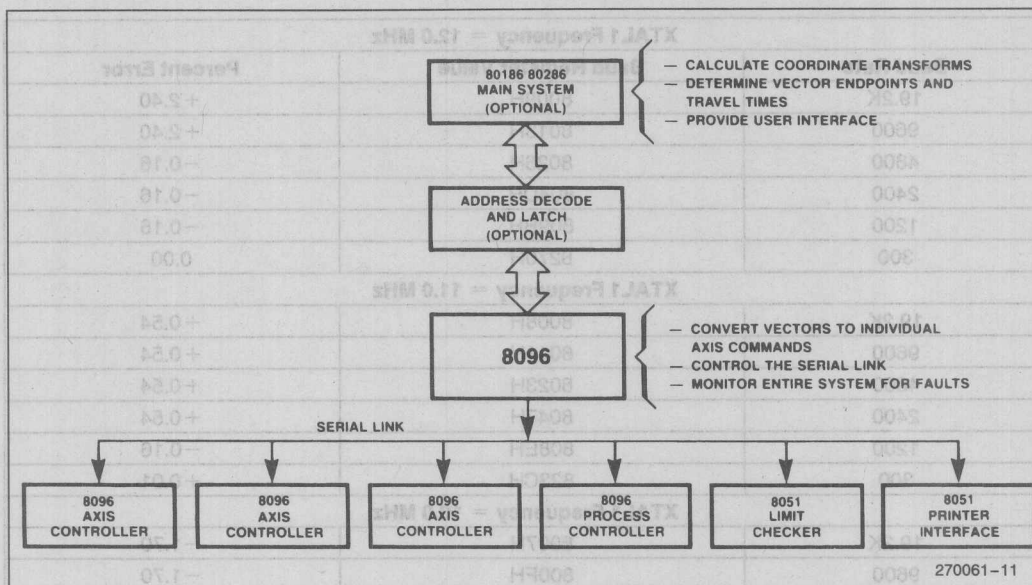


Figure 2-18. Multiprocessor Communication

Mode 0, the synchronous mode, is typically used for interfacing to shift registers for I/O expansion. The software to control this mode involves the REN (Receiver ENable) bit, the clearing of the RI bit, and writing to SBUF. To transmit to a shift register, REN is set to zero and SBUF is loaded with the information. The information will be sent and then the TI flag will be set. There are two ways to cause a reception to begin. The first is by causing a rising edge to occur on the REN bit, the second is by clearing RI with REN = 1. In either case, RI is set again when the received byte is available in SBUF.

2.3.5. A to D CONVERTER

Analog inputs are frequently required in a microcontroller application. The 8097 has a 10-bit A to D converter that can use any one of eight input channels. The conversions are done using the successive approximation method, and require 168 state times (42 microseconds with a 12 MHz clock.)

The results are guaranteed monotonic by design of the converter. This means that if the analog input voltage changes, even slightly, the digital value will either stay the same or change in the same direction as the analog

input. When doing process control algorithms, it is frequently the changes in inputs that are required, not the absolute accuracy of the value. For this reason, even if the absolute accuracy of a 10-bit converter is the same as that of an 8-bit converter, the 10-bit monotonic converter is much more useful.

Since most of the analog inputs which are monitored by a microcontroller change very slowly relative to the 42 microsecond conversion time, it is acceptable to use a capacitive filter on each input instead of a sample and hold. The 8097 does not have an internal sample and hold, so it is necessary to ensure that the input signal does not change during the conversion time. The input to the A/D must be between ANGND and VREF. ANGND must be within a few millivolts of VSS and VREF must be within a few tenths of a volt of VCC.

Using the A to D converter on the 8097 can be a very low software overhead task because of the interrupt and HSO unit structure. The A to D can be started by the HSO unit at a preset time. When the conversion is complete it is possible to generate an interrupt. By using these features the A to D can be run under complete interrupt control. The A to D can also be directly

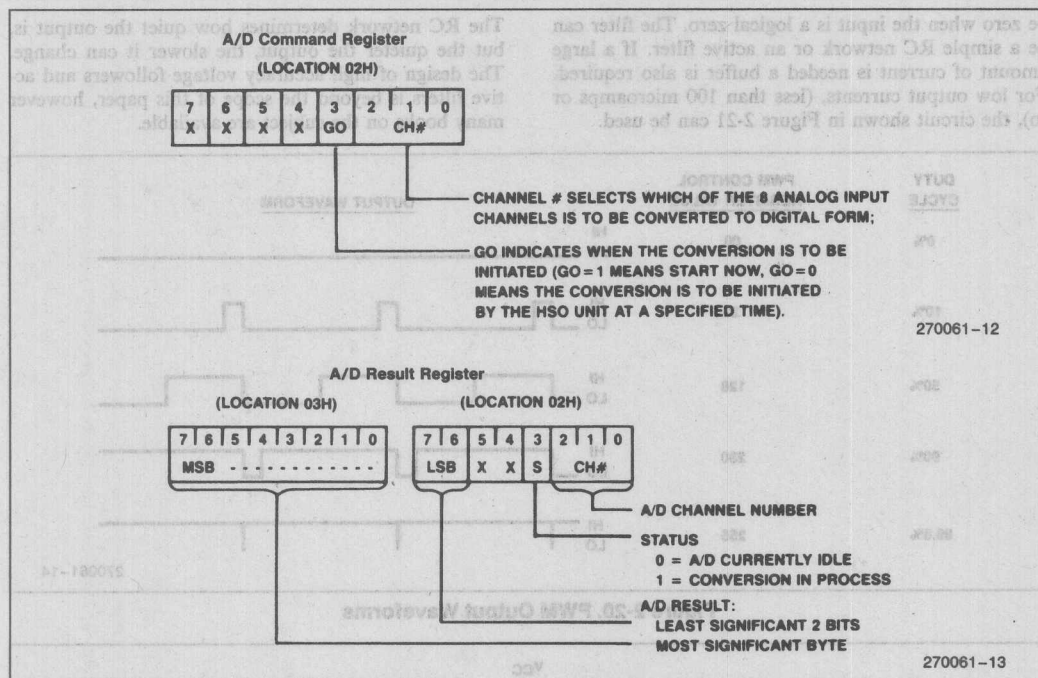


Figure 2-19. A to D Result/Command Register

controlled by software flags which are located in the AD_RESULT/AD_COMMAND Register, shown in Figure 2-19.

2.3.6. PWM REGISTER

Analog outputs are just as important as analog inputs when connecting to a piece of equipment. True digital to analog converters are difficult to make on a microprocessor because of all of the digital noise and the necessity of providing an on chip, relatively high current, rail to rail driver. They also take up a fair amount of silicon area which can be better used for other features. The A to D converter does use a D to A, but the currents involved are very small.

For many applications an analog output signal can be replaced by a Pulse Width Modulated (PWM) signal. This signal can be easily generated in hardware, and

takes up much less silicon area than a true D to A. The signal is a variable duty cycle, fixed frequency waveform that can be integrated to provide an approximation to an analog output. The frequency is fixed at a period of 64 microseconds for a 12 MHz clock speed. Controlling the PWM simply requires writing the desired duty cycle value (an 8-bit value) to the PWM Register. Some typical output waveforms that can be generated are shown in Figure 2-20.

Converting the PWM signal to an analog signal varies in difficulty, depending upon the requirements of the system. Some systems, such as motors or switching power supplies actually require a PWM signal, not a true analog one. For many other cases it is necessary only to amplify the signal so that it switches rail-to-rail, and then filter it. Switching rail-to-rail means that the output of the amplifier will be a reference value when the input is a logical one, and the output will

be zero when the input is a logical zero. The filter can be a simple RC network or an active filter. If a large amount of current is needed a buffer is also required. For low output currents, (less than 100 microamps or so), the circuit shown in Figure 2-21 can be used.

The RC network determines how quiet the output is, but the quieter the output, the slower it can change. The design of high accuracy voltage followers and active filters is beyond the scope of this paper, however many books on the subject are available.

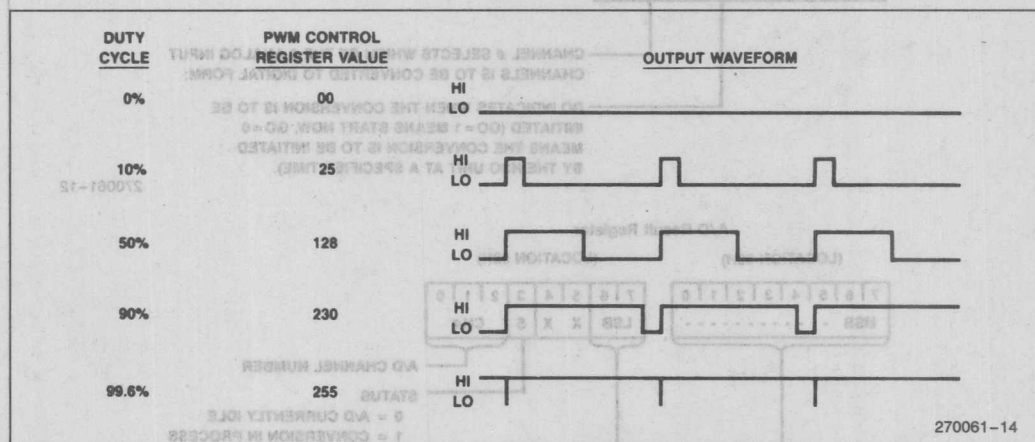


Figure 2-20. PWM Output Waveforms

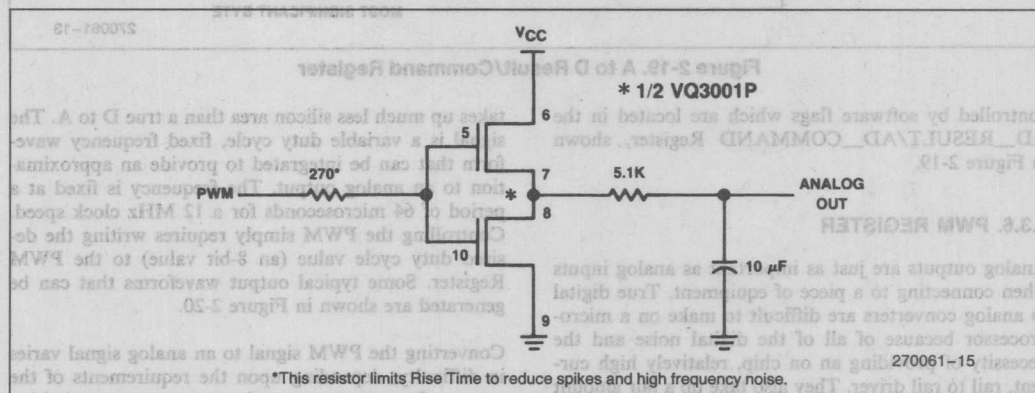


Figure 2-21. PWM to Analog Conversion Circuitry

3.0 BASIC SOFTWARE EXAMPLES

The examples in this section show how to use each I/O feature individually. Examples of using more than one feature at a time are described in section 4. All of the examples in this ap-note are set up to be used as listed. If run through ASM96 they will load and run on an SBE-96. In order to insure that the programs work, the stack pointer is initialized at the beginning of each program. If the programs are going to be used as modules of other programs, the stack pointer initialization should only be used at the beginning of the main program.

To avoid repetitive declarations the "include" file "DEMO96.INC", shown in Listing 3-1, is used. ASM-96 will insert this file into the code file whenever the directive "INCLUDE DEMO96.INC" is used. The file contains the definitions for the SFRs and other variables. The include statement has been placed in all of the examples. It should be noted that some of the lab-

els in this file are different from those in the file 8096.INC that is provided in the ASM-96 package.

3.1. Using the 8096's Processing Section

3.1.1. TABLE INTERPOLATION

A good way of increasing speed for many processing tasks is to use table lookup with interpolation. This can eliminate lengthy calculations in many algorithms. Frequently it is used in programs that generate sine waveforms, use exponents in calculations, or require some non-linear function of a given input variable. Table lookup can also be used without interpolation to determine the output state of I/O devices for a given state of a set of input devices. The procedure is also a good example of 8096 code as it uses many of the software features. Two ways of making a lookup table are described, one way uses more calculation time, the second way uses more table space.

; *****			
; DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096			
; *****			
ZERO	EQU	00h:WORD	; R/W
AD_COMMAND	EQU	02h:BYTE	; W
AD_RESULT_LO	EQU	02h:BYTE	; R
AD_RESULT_HI	EQU	03h:BYTE	; R
HSI_MODE	EQU	03h:BYTE	; W
HSI_TIME	EQU	04h:WORD	; W
HSI_TIME	EQU	04h:WORD	; R
HSI_COMMAND	EQU	06h:BYTE	; W
HSI_STATUS	EQU	06h:BYTE	; R
SBUF	EQU	07h:BYTE	; R/W
INT_MASK	EQU	08h:BYTE	; R/W
INT_PENDING	EQU	09h:BYTE	; R/W
SPCON	EQU	11h:BYTE	
SPSTAT	EQU	11h:BYTE	
WATCHDOG	EQU	0Ah:BYTE	; W WATCHDOG TIMER
TIMER1	EQU	0Ah:WORD	; R
TIMER2	EQU	0Ch:WORD	; R
PORT0	EQU	0Eh:BYTE	; R
BAUD_REG	EQU	0Eh:BYTE	; W
PORT1	EQU	0Fh:BYTE	; R/W
PORT2	EQU	10h:BYTE	; R/W
IOC0	EQU	15h:BYTE	; W
IOS0	EQU	15h:BYTE	; R
IOC1	EQU	16h:BYTE	; W
IOS1	EQU	16h:BYTE	; R
PWM_CONTROL	EQU	17h:BYTE	; W
SP	EQU	18h:WORD	; R/W STACK POINTER

RSEG at lch			
AX:	DSW	1	
DX:	DSW	1	
BX:	DSW	1	
CX:	DSW	1	
AL	EQU	AX	:BYTE
AH	EQU	(AX+1)	:BYTE

Listing 3-1. Include File DEMO.96.INC

In both methods the procedure is similar. Values of a function are stored in memory for specific input values. To compute the output function for an input that is not listed, a linear approximation is made based on the nearest inputs and nearest outputs. As an example, consider the table below.

If the input value was one of those listed then there would be no problem. Unfortunately the real world is never so kind. The input number will probably be 259 or something similar. If this is the case linear interpolation would provide a reasonable result. The formula is:

$$\text{Delta Out} = \frac{\text{Upper Output} - \text{Lower Output}}{\text{Upper Input} - \text{Lower Input}} * (\text{Actual Input} - \text{Lower Input})$$

$$\text{Actual Output} = \text{Lower Output} + \text{Delta Out}$$

For the value of 259 the solution is:

$$\text{Delta Out} = \frac{900 - 400}{300 - 200} * (259 - 200) = \frac{500}{100} * 59 = 5 * 59 = 295$$

$$\text{Actual Output} = 400 + 295 = 695$$

To make the algorithm easier, (and therefore faster), it is appropriate to limit the range and accuracy of the function to only what is needed. It is also advantageous to make the input step (Upper Input-Lower Input) equal to a power of 2. This allows the substitution of multiple right shifts for a divide operation, thus speeding up throughput. The 8096 allows multiple arithmetic right shifts with a single instruction providing a very fast divide if the divisor is a power of two.

For the purpose of an example, a program with a 12-bit output and an 8-bit input has been written. An input step of 16 (2**4) was selected. To cover the input range 17 words are needed, 255/16 + 1 word to handle values in the last 15 bytes of input range. Although only 12 bits are required for the output, the 16-bit architecture offers no penalty for using 16 instead of 12 bits.

The program for this example, shown in Listing 3-2, uses the definitions and equates from Listing 3-1, only the additional equates and definitions are shown in the code.

Input Value	Relative Table Address	Table Value
100	0001H	100
200	0002H	400
300	0003H	900
400	0004H	1600


```

$TITLE('INTER1.APT: Interpolation routine 1')
; ; ; ; ; 8096 Assembly code for table lookup and interpolation

$INCLUDE(;;F1:DEMO96.INC) ; Include demo definitions

RSEG at 22H

IN_VAL:      ddb 1 ; Actual Input Value
TABLE_LOW:   ddb 1
TABLE_HIGH:  ddb 1
IN_DIF:      ddb 1 ; Upper Input - Lower Input
IN_DIFB:     equ IN_DIF, 1 byte
TAB_DIF:     ddb 1 ; Upper Output - Lower Output
OUT:         ddb 1
RESULT:      ddb 1
OUT_DIF:     ddb 1 ; Delta Out

CSEG at 2080H

LD SP, $100H
  
```

Listing 3-2. ASM-96 Code for Table Lookup Routine 1

```

look:  LDB     AL, IN_VAL      ; Load temp with Actual Value
       SHRB   AL, #3         ; Divide the byte by 8
       ANDB   AL, #1111110B  ; Insure AL is a word address
                               ; This effectively divides AL by 2
                               ; so AL = IN_VAL/16

       LDBZ   AX, AL         ; Load byte AL to word AX
       LD     TABLE_LOW, TABLE [AX] ; TABLE_LOW is loaded with the value
                                   ; in the table at table location AX

       LD     TABLE_HIGH, (TABLE+2)[AX] ; TABLE_HIGH is loaded with the
                                   ; value in the table at table
                                   ; location AX+2
                                   ; (The next value in the table)

       SUB     TAB_DIF, TABLE_HIGH, TABLE_LOW
                                   ; TAB_DIF=TABLE_HIGH-TABLE_LOW

       ANDB   IN_DIFB, IN_VAL, #0FH ; IN_DIFB=least significant 4 bits
                                   ; of IN_VAL

       LDBZ   IN_DIF, IN_DIFB ; Load byte IN_DIFB to word IN_DIF

       MUL     OUT_DIF, IN_DIF, TAB_DIF
                                   ; Output difference =
                                   ; Input_difference*Table_difference

       SHRAL   OUT_DIF, #4      ; Divide by 16 (2*4)

       ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                                   ; generated with truncated IN_VAL
                                   ; as input

       SHRA    OUT, #4         ; Round to 12-bit answer

       ADDC    OUT, zero       ; Round up if Carry = 1

no_inc: ST     OUT, RESULT      ; Store OUT to RESULT

       BR     look            ; Branch to "look:"

cseg   AT 2100H

table: DCW     0000H, 2000H, 3400H, 4C00H ; A random function
       DCW     5D00H, 6A00H, 7200H, 7800H
       DCW     7B00H, 7D00H, 7600H, 6D00H
       DCW     5D00H, 4B00H, 3400H, 2200H
       DCW     1000H

END

```

270061-18

Listing 3-2. ASM-96 Code for Table Lookup Routine 1 (Continued)

If the function is known at the time of writing the software it is also possible to calculate in advance the change in the output function for a given change in the input. This method can save a divide and a few other instructions at the expense of doubling the size of the

lookup table. There are many applications where time is critical and code space is overly abundant. In these cases the code in Listing 3-3 will work to the same specifications as the previous example.

6

```

$TITLE('INTER2.APT: Interpolation routine 2')

; ; ; ; ; 8096 Assembly code for table lookup and interpolation
; ; ; ; ; Using tabled values in place of division

$INCLUDE('F1:DEMO96.INC') ; Include demo definitions

RSEG at 24H

IN_VAL:      dsb      1      ; Actual Input Value
TABLE_LOW:   dsb      1      ; Table value for function
TABLE_INC:   dsb      1      ; Incremental change in function
IN_DIFB:     dsb      1      ; Upper Input - Lower Input
IN_DIF:      equ      IN_DIF ; byte
OUT:         dsb      1
RESULT:      dsb      1
OUT_DIF:     dsb      1      ; Delta Out

```

270061-19

Listing 3-3. ASM-96 Code For Table Lookup Routine 2

```

LD      SP, #100H      ; Initialize SP to top of reg. file

look:   LDB      AL, IN_VAL      ; Load temp with Actual Value
        SHRB     AL, #3         ; Divide the byte by 8
        ANDB     AL, #1111110B ; Divide the word address by 2
        LDBZ     AX, AL         ; so AL = IN_VAL/16
        LDBZ     AX, AL         ; Load byte AL to word AX

LD      TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value
                                   ; in the value table at location AX

LD      TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value
                                   ; in the increment table at
                                   ; location AX

ANDB     IN_DIFB, IN_VAL, #0FH ; IN_DIFB=least significant 4 bits
                                   ; of IN_VAL
LDBZ     IN_DIF, IN_DIFB        ; Load byte IN_DIFB to word IN_DIF

MUL      OUT_DIF, IN_DIF, TABLE_INC ; Output difference =
                                   ; Input_difference*Incremental_change

ADD      OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                                   ; generated with truncated IN_VAL
                                   ; as input
SHR      OUT, #4               ; Round to 12-bit answer
ADDC     OUT, zero             ; Round up if Carry = 1

no_inc: ST      OUT, RESULT      ; Store OUT to RESULT
        BR      look           ; Branch to "look:"

cseg     AT 2100H

val_table:
DCW      0000H, 2000H, 3400H, 4C00H ; A random function
DCW      5D00H, 6A00H, 7200H, 7800H
DCW      7B00H, 7D00H, 7600H, 6D00H
DCW      5D00H, 4B00H, 3400H, 2200H
DCW      1000H

inc_table:
DCW      0200H, 0140H, 0180H, 0110H ; Table of incremental
DCW      00D0H, 0080H, 0060H, 0030H ; differences
DCW      00020H, 0FF90H, 0FF70H, 0FF00H
DCW      0FE0H, 0FE90H, 0FEE0H, 0FEE0H

```

END

270061-20

Listing 3-3. ASM-96 Code for Table Lookup Routine 2 (Continued)

By making use of the second lookup table, one word of RAM was saved and 16 state times. In most cases this time savings would not make much of a difference, but when pushing the processor to the limit, microseconds can make or break a design.

3.1.2. PL/M-96

Intel provides high level language support for most of its micro processors and microcontrollers in the form of PL/M. Specifically, PL/M refers to a family of languages, each similar in syntax, but specialized for the device for which it generates code. The PL/M syntax is similar to PL/1, and is easy to learn. PLM-96 is the version of PL/M used for the 8096. It is very code efficient as it was written specifically for the MCS-96 family. PLM-96 most closely resembles PLM-86, although it has bit and I/O functions similar to PLM-51. One line of PL/M-code can take the place of many

lines of assembly code. This is advantageous to the programmer, since code can usually be written at a set number of lines per hour, so the less lines of code that need to be written, the faster the task can be completed.

If the first example of interpolation is considered, the PLM-96 code would be written as shown in Listing 3-4. Note that version 1.0 of PLM-96 does not support 32-bit results of 16 by 16 multiplies, so the ASM-96 procedure "DMPY" is used. Procedure DMPY, shown in Listing 3-5, must be assembled and linked with the compiled PLM-96 program using RL-96, the relocater and linker. The command line to be used is:

RL96 PLMEX1.OBJ, DMPY.OBJ, PLM96.LIB &
to PLMOUT.OBJ ROM (2080H-3FFFH)


```

/* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION */
PLMEX:    DO;

DECLARE IN_VAL    WORD PUBLIC;
DECLARE TABLE_LOW    INTEGER PUBLIC;
DECLARE TABLE_HIGH    INTEGER PUBLIC;
DECLARE TABLE_DIP    INTEGER PUBLIC;
DECLARE OUT        INTEGER PUBLIC;
DECLARE RESULT    INTEGER PUBLIC;
DECLARE OUT_DIP    LONGINT PUBLIC;
DECLARE TEMP        WORD PUBLIC;

DECLARE TABLE(17)    INTEGER DATA (
    0000H, 2000H, 3400H, 4C00H, /* A random function */
    5D00H, 6A00H, 7200H, 7800H,
    7B00H, 7D00H, 7600H, 6D00H,
    5D00H, 4B00H, 3400H, 2200H,
    1000H);

DMPY:    PROCEDURE (A,B) LONGINT EXTERNAL;
DECLARE (A,B) INTEGER;
END DMPY;

LOOP:
TEMP=SHR(IN_VAL,4); /* TEMP is the most significant 4 bits of IN_VAL */
TABLE_LOW=TABLE(TEMP); /* If "TEMP" was replaced by "SHR(IN_VAL,4)" */
TABLE_HIGH=TABLE(TEMP+1); /* The code would work but the 8096 would */
/* do two shifts */

TABLE_DIP=TABLE_HIGH-TABLE_LOW;
OUT_DIP=DMPY(TABLE_DIP,SIGNED(IN_VAL AND 0FH)) /16;

OUT=SAR((TABLE_LOW+OUT_DIP),4); /* SAR performs an arithmetic right shift,
/* in this case 4 places are shifted */

IF CARRY=0 THEN RESULT=OUT; /* Using the hardware flags must be done */
ELSE RESULT=OUT+1; /* with care to ensure the flag is tested */
/* in the desired instruction sequence */

GOTO LOOP;

/* END OF PLM-96 CODE */
END;

```

270061-21

Listing 3-4. PLM-96 Code For Table Lookup Routine 1

```

$TITLE('MULT.APT: 16*16 multiply procedure for PLM-96')
SP    EQU    18H:word

rseg    EXTRN    PLMREG:long

cseg
PUBLIC DMPY    ; Multiply two integers and return a
                ; longint result in AX, DX registers

DMPY:    POP     PLMREG+4    ; Load return address
POP     PLMREG    ; Load one operand
MUL     PLMREG,[SP]+    ; Load second operand and increment SP

BR     [PLMREG+4]    ; Return to PLM code.

END

```

270061-22

Listing 3-5. 32-Bit Result Multiply Procedure For PLM-96

Using PLM, code requires less lines, is much faster to write, and easier to maintain, but may take slightly longer to run. For this example, the assembly code generated by the PLM-96 compiler takes 56.75 microseconds to run instead of 30.75 microseconds. If PLM-96 performed the 32-bit result multiply instead of using the ASM-96 routine the PLM code would take 41.5 microseconds to run. The actual code listings are shown in Appendix A.

3.2. Using the I/O Section

3.2.1. USING THE HSI UNIT

One of the most frequent uses of the HSI is to measure the time between events. This can be used for frequency determination in lab instruments, or speed/acceleration information when connected to pulse type encoders. The code in Listing 3-6 can be used to determine the high and low times of the signals on two lines. This code can be easily expanded to 4 lines and can also be modified to work as an interrupt routine.

Frequently it is also desired to keep track of the number of events which have occurred, as well as how often they are occurring. By using a software counter this feature can be added to the above code. This code depends on the software responding to the change in line state before the line changes again. If this cannot be guaranteed then it may be necessary to use 2 HSI lines for each incoming line. In this case one HSI line would look for falling edges while the other looks for rising edges. The code in Listing 3-7 includes both the counter feature and the edge detect feature.

The uses for this type of routine are almost endless. In instrumentation it can be used to determine frequency on input lines, or perhaps baud rate for a self adjusting serial port. Section 4.2 contains an example of making a software serial port using the HSI unit. Interfacing to some form of mechanically generated position information is a very frequent use of the HSI. The applications in this category include motor control, precise positioning (print heads, disk drives, etc.), engine control and

```

$TITLE('PULSE.APT: Measuring pulses using the HSI unit')
$INCLUDE(DEMO96.INC)
rseg at 28H
HIGH_TIME: dsw 1
LOW_TIME: dsw 1
PERIOD: dsw 1
HI_EDGE: dsw 1
LO_EDGE: dsw 1

cseg at 2080H
LD SP, #100H
LDB IOC0, #00000001B ; Enable HSI 0
LDB HSI_MODE, #00001111B ; HSI 0 look for either edge

wait: ADD PERIOD, HIGH_TIME, LOW_TIME
JBS IOS1, 6, contin ; If FIFO is full
JBC IOS1, 7, wait ; Wait while no pulse is entered

contin: LDB AL, HSI_STATUS ; Load status; Note that reading
; HSI_TIME clears HSI_STATUS
LD BX, HSI_TIME ; Load the HSI_TIME
JBS AL, 1, hsi_hi ; Jump if HSI.0 is high

hsi_lo: ST BX, LO_EDGE
SUB HIGH_TIME, LO_EDGE, HI_EDGE
BR wait

hsi_hi: ST BX, HI_EDGE
SUB LOW_TIME, HI_EDGE, LO_EDGE
BR wait

END

```

270061-23

Listing 3-6. Measuring Pulses Using The HSI Unit

transmission control. The HSI unit is used extensively in the example in section 4.3.

3.2.2. USING THE HSO UNIT

Although the HSO has many uses, the best example is that of a multiple PWM output. This program, shown in Listing 3-8, is simple enough to be easily understood, yet it shows how to use the HSO for a task which can be complex. In order for this program to operate, another program needs to set up the on and off time variables for each line. The program also requires that a

HSO line not change so quickly that it changes twice between consecutive reads of I/O Status Register 0, (IOS0).

A very eye catching example can be made by having the program output waveforms that vary over time. The driver routine in Listing 3-10 can be linked to the above program to provide this function. Linking is accomplished using RL96, the relocatable linker for the 8096. Information for using RL96 can be found in the "MCS-96 Utilities Users Guide", listed in the bibliography. In order for the program to link, the register dec-

```

$TITLE ('ENHSI.APT: ENHANCED HSI PULSE ROUTINE')
$INCLUDE (DEMO96.INC)

RSEG AT 28H

TIME: DSW 1
LAST_RISE: DSW 1
LAST_FALL: DSW 1
HSI_S0: DSB 1
IOS1_BAK: DSB 1
PERIOD: DSW 1
LOW_TIME: DSW 1
HIGH_TIME: DSW 1
COUNT: DSW 1

;-----
; 2080H
; 2081H
; 2082H
; 2083H
; 2084H
; 2085H
; 2086H
; 2087H
; 2088H
; 2089H
; 208AH
; 208BH
; 208CH
; 208DH
; 208EH
; 208FH
; 2090H
; 2091H
; 2092H
; 2093H
; 2094H
; 2095H
; 2096H
; 2097H
; 2098H
; 2099H
; 209AH
; 209BH
; 209CH
; 209DH
; 209EH
; 209FH
; 20A0H
; 20A1H
; 20A2H
; 20A3H
; 20A4H
; 20A5H
; 20A6H
; 20A7H
; 20A8H
; 20A9H
; 20AAH
; 20ABH
; 20ACH
; 20ADH
; 20AEH
; 20AFH
; 20B0H
; 20B1H
; 20B2H
; 20B3H
; 20B4H
; 20B5H
; 20B6H
; 20B7H
; 20B8H
; 20B9H
; 20BAH
; 20BBH
; 20BCH
; 20BDH
; 20BEH
; 20BFH
; 20C0H
; 20C1H
; 20C2H
; 20C3H
; 20C4H
; 20C5H
; 20C6H
; 20C7H
; 20C8H
; 20C9H
; 20CAH
; 20CBH
; 20CEH
; 20CFH
; 20D0H
; 20D1H
; 20D2H
; 20D3H
; 20D4H
; 20D5H
; 20D6H
; 20D7H
; 20D8H
; 20D9H
; 20DAH
; 20DBH
; 20DCH
; 20DDH
; 20DEH
; 20DFH
; 20E0H
; 20E1H
; 20E2H
; 20E3H
; 20E4H
; 20E5H
; 20E6H
; 20E7H
; 20E8H
; 20E9H
; 20EAH
; 20EBH
; 20ECH
; 20EDH
; 20EEH
; 20EFH
; 20F0H
; 20F1H
; 20F2H
; 20F3H
; 20F4H
; 20F5H
; 20F6H
; 20F7H
; 20F8H
; 20F9H
; 20FAH
; 20FBH
; 20FCH
; 20FDH
; 20FEH
; 20FFH
; 2100H
; 2101H
; 2102H
; 2103H
; 2104H
; 2105H
; 2106H
; 2107H
; 2108H
; 2109H
; 210AH
; 210BH
; 210CH
; 210DH
; 210EH
; 210FH
; 2110H
; 2111H
; 2112H
; 2113H
; 2114H
; 2115H
; 2116H
; 2117H
; 2118H
; 2119H
; 211AH
; 211BH
; 211CH
; 211DH
; 211EH
; 211FH
; 2120H
; 2121H
; 2122H
; 2123H
; 2124H
; 2125H
; 2126H
; 2127H
; 2128H
; 2129H
; 212AH
; 212BH
; 212CH
; 212DH
; 212EH
; 212FH
; 2130H
; 2131H
; 2132H
; 2133H
; 2134H
; 2135H
; 2136H
; 2137H
; 2138H
; 2139H
; 213AH
; 213BH
; 213CH
; 213DH
; 213EH
; 213FH
; 2140H
; 2141H
; 2142H
; 2143H
; 2144H
; 2145H
; 2146H
; 2147H
; 2148H
; 2149H
; 214AH
; 214BH
; 214CH
; 214DH
; 214EH
; 214FH
; 2150H
; 2151H
; 2152H
; 2153H
; 2154H
; 2155H
; 2156H
; 2157H
; 2158H
; 2159H
; 215AH
; 215BH
; 215CH
; 215DH
; 215EH
; 215FH
; 2160H
; 2161H
; 2162H
; 2163H
; 2164H
; 2165H
; 2166H
; 2167H
; 2168H
; 2169H
; 216AH
; 216BH
; 216CH
; 216DH
; 216EH
; 216FH
; 2170H
; 2171H
; 2172H
; 2173H
; 2174H
; 2175H
; 2176H
; 2177H
; 2178H
; 2179H
; 217AH
; 217BH
; 217CH
; 217DH
; 217EH
; 217FH
; 2180H
; 2181H
; 2182H
; 2183H
; 2184H
; 2185H
; 2186H
; 2187H
; 2188H
; 2189H
; 218AH
; 218BH
; 218CH
; 218DH
; 218EH
; 218FH
; 2190H
; 2191H
; 2192H
; 2193H
; 2194H
; 2195H
; 2196H
; 2197H
; 2198H
; 2199H
; 219AH
; 219BH
; 219CH
; 219DH
; 219EH
; 219FH
; 21A0H
; 21A1H
; 21A2H
; 21A3H
; 21A4H
; 21A5H
; 21A6H
; 21A7H
; 21A8H
; 21A9H
; 21AAH
; 21ABH
; 21ACH
; 21ADH
; 21AEH
; 21AFH
; 21B0H
; 21B1H
; 21B2H
; 21B3H
; 21B4H
; 21B5H
; 21B6H
; 21B7H
; 21B8H
; 21B9H
; 21BAH
; 21BBH
; 21BCH
; 21BDH
; 21BEH
; 21BFH
; 21C0H
; 21C1H
; 21C2H
; 21C3H
; 21C4H
; 21C5H
; 21C6H
; 21C7H
; 21C8H
; 21C9H
; 21CAH
; 21CBH
; 21CEH
; 21CFH
; 21D0H
; 21D1H
; 21D2H
; 21D3H
; 21D4H
; 21D5H
; 21D6H
; 21D7H
; 21D8H
; 21D9H
; 21DAH
; 21DBH
; 21DCH
; 21DDH
; 21DEH
; 21DFH
; 21E0H
; 21E1H
; 21E2H
; 21E3H
; 21E4H
; 21E5H
; 21E6H
; 21E7H
; 21E8H
; 21E9H
; 21EAH
; 21EBH
; 21ECH
; 21EDH
; 21EEH
; 21EFH
; 21F0H
; 21F1H
; 21F2H
; 21F3H
; 21F4H
; 21F5H
; 21F6H
; 21F7H
; 21F8H
; 21F9H
; 21FAH
; 21FBH
; 21FCH
; 21FDH
; 21FEH
; 21FFH
; 2200H
; 2201H
; 2202H
; 2203H
; 2204H
; 2205H
; 2206H
; 2207H
; 2208H
; 2209H
; 220AH
; 220BH
; 220CH
; 220DH
; 220EH
; 220FH
; 2210H
; 2211H
; 2212H
; 2213H
; 2214H
; 2215H
; 2216H
; 2217H
; 2218H
; 2219H
; 221AH
; 221BH
; 221CH
; 221DH
; 221EH
; 221FH
; 2220H
; 2221H
; 2222H
; 2223H
; 2224H
; 2225H
; 2226H
; 2227H
; 2228H
; 2229H
; 222AH
; 222BH
; 222CH
; 222DH
; 222EH
; 222FH
; 2230H
; 2231H
; 2232H
; 2233H
; 2234H
; 2235H
; 2236H
; 2237H
; 2238H
; 2239H
; 223AH
; 223BH
; 223CH
; 223DH
; 223EH
; 223FH
; 2240H
; 2241H
; 2242H
; 2243H
; 2244H
; 2245H
; 2246H
; 2247H
; 2248H
; 2249H
; 224AH
; 224BH
; 224CH
; 224DH
; 224EH
; 224FH
; 2250H
; 2251H
; 2252H
; 2253H
; 2254H
; 2255H
; 2256H
; 2257H
; 2258H
; 2259H
; 225AH
; 225BH
; 225CH
; 225DH
; 225EH
; 225FH
; 2260H
; 2261H
; 2262H
; 2263H
; 2264H
; 2265H
; 2266H
; 2267H
; 2268H
; 2269H
; 226AH
; 226BH
; 226CH
; 226DH
; 226EH
; 226FH
; 2270H
; 2271H
; 2272H
; 2273H
; 2274H
; 2275H
; 2276H
; 2277H
; 2278H
; 2279H
; 227AH
; 227BH
; 227CH
; 227DH
; 227EH
; 227FH
; 2280H
; 2281H
; 2282H
; 2283H
; 2284H
; 2285H
; 2286H
; 2287H
; 2288H
; 2289H
; 228AH
; 228BH
; 228CH
; 228DH
; 228EH
; 228FH
; 2290H
; 2291H
; 2292H
; 2293H
; 2294H
; 2295H
; 2296H
; 2297H
; 2298H
; 2299H
; 229AH
; 229BH
; 229CH
; 229DH
; 229EH
; 229FH
; 22A0H
; 22A1H
; 22A2H
; 22A3H
; 22A4H
; 22A5H
; 22A6H
; 22A7H
; 22A8H
; 22A9H
; 22AAH
; 22ABH
; 22ACH
; 22ADH
; 22AEH
; 22AFH
; 22B0H
; 22B1H
; 22B2H
; 22B3H
; 22B4H
; 22B5H
; 22B6H
; 22B7H
; 22B8H
; 22B9H
; 22BAH
; 22BBH
; 22BCH
; 22BDH
; 22BEH
; 22BFH
; 22C0H
; 22C1H
; 22C2H
; 22C3H
; 22C4H
; 22C5H
; 22C6H
; 22C7H
; 22C8H
; 22C9H
; 22CAH
; 22CBH
; 22CEH
; 22CFH
; 22D0H
; 22D1H
; 22D2H
; 22D3H
; 22D4H
; 22D5H
; 22D6H
; 22D7H
; 22D8H
; 22D9H
; 22DAH
; 22DBH
; 22DCH
; 22DDH
; 22DEH
; 22DFH
; 22E0H
; 22E1H
; 22E2H
; 22E3H
; 22E4H
; 22E5H
; 22E6H
; 22E7H
; 22E8H
; 22E9H
; 22EAH
; 22EBH
; 22ECH
; 22EDH
; 22EEH
; 22EFH
; 22F0H
; 22F1H
; 22F2H
; 22F3H
; 22F4H
; 22F5H
; 22F6H
; 22F7H
; 22F8H
; 22F9H
; 22FAH
; 22FBH
; 22FCH
; 22FDH
; 22FEH
; 22FFH
; 2300H
; 2301H
; 2302H
; 2303H
; 2304H
; 2305H
; 2306H
; 2307H
; 2308H
; 2309H
; 230AH
; 230BH
; 230CH
; 230DH
; 230EH
; 230FH
; 2310H
; 2311H
; 2312H
; 2313H
; 2314H
; 2315H
; 2316H
; 2317H
; 2318H
; 2319H
; 231AH
; 231BH
; 231CH
; 231DH
; 231EH
; 231FH
; 2320H
; 2321H
; 2322H
; 2323H
; 2324H
; 2325H
; 2326H
; 2327H
; 2328H
; 2329H
; 232AH
; 232BH
; 232CH
; 232DH
; 232EH
; 232FH
; 2330H
; 2331H
; 2332H
; 2333H
; 2334H
; 2335H
; 2336H
; 2337H
; 2338H
; 2339H
; 233AH
; 233BH
; 233CH
; 233DH
; 233EH
; 233FH
; 2340H
; 2341H
; 2342H
; 2343H
; 2344H
; 2345H
; 2346H
; 2347H
; 2348H
; 2349H
; 234AH
; 234BH
; 234CH
; 234DH
; 234EH
; 234FH
; 2350H
; 2351H
; 2352H
; 2353H
; 2354H
; 2355H
; 2356H
; 2357H
; 2358H
; 2359H
; 235AH
; 235BH
; 235CH
; 235DH
; 235EH
; 235FH
; 2360H
; 2361H
; 2362H
; 2363H
; 2364H
; 2365H
; 2366H
; 2367H
; 2368H
; 2369H
; 236AH
; 236BH
; 236CH
; 236DH
; 236EH
; 236FH
; 2370H
; 2371H
; 2372H
; 2373H
; 2374H
; 2375H
; 2376H
; 2377H
; 2378H
; 2379H
; 237AH
; 237BH
; 237CH
; 237DH
; 237EH
; 237FH
; 2380H
; 2381H
; 2382H
; 2383H
; 2384H
; 2385H
; 2386H
; 2387H
; 2388H
; 2389H
; 238AH
; 238BH
; 238CH
; 238DH
; 238EH
; 238FH
; 2390H
; 2391H
; 2392H
; 2393H
; 2394H
; 2395H
; 2396H
; 2397H
; 2398H
; 2399H
; 239AH
; 239BH
; 239CH
; 239DH
; 239EH
; 239FH
; 23A0H
; 23A1H
; 23A2H
; 23A3H
; 23A4H
; 23A5H
; 23A6H
; 23A7H
; 23A8H
; 23A9H
; 23AAH
; 23ABH
; 23ACH
; 23ADH
; 23AEH
; 23AFH
; 23B0H
; 23B1H
; 23B2H
; 23B3H
; 23B4H
; 23B5H
; 23B6H
; 23B7H
; 23B8H
; 23B9H
; 23BAH
; 23BBH
; 23BCH
; 23BDH
; 23BEH
; 23BFH
; 23C0H
; 23C1H
; 23C2H
; 23C3H
; 23C4H
; 23C5H
; 23C6H
; 23C7H
; 23C8H
; 23C9H
; 23CAH
; 23CBH
; 23CEH
; 23CFH
; 23D0H
; 23D1H
; 23D2H
; 23D3H
; 23D4H
; 23D5H
; 23D6H
; 23D7H
; 23D8H
; 23D9H
; 23DAH
; 23DBH
; 23DCH
; 23DDH
; 23DEH
; 23DFH
; 23E0H
; 23E1H
; 23E2H
; 23E3H
; 23E4H
; 23E5H
; 23E6H
; 23E7H
; 23E8H
; 23E9H
; 23EAH
; 23EBH
; 23ECH
; 23EDH
; 23EEH
; 23EFH
; 23F0H
; 23F1H
; 23F2H
; 23F3H
; 23F4H
; 23F5H
; 23F6H
; 23F7H
; 23F8H
; 23F9H
; 23FAH
; 23FBH
; 23FCH
; 23FDH
; 23FEH
; 23FFH
; 2400H
; 2401H
; 2402H
; 2403H
; 2404H
; 2405H
; 2406H
; 2407H
; 2408H
; 2409H
; 240AH
; 240BH
; 240CH
; 240DH
; 240EH
; 240FH
; 2410H
; 2411H
; 2412H
; 2413H
; 2414H
; 2415H
; 2416H
; 2417H
; 2418H
; 2419H
; 241AH
; 241BH
; 241CH
; 241DH
; 241EH
; 241FH
; 2420H
; 2421H
; 2422H
; 2423H
; 2424H
; 2425H
; 2426H
; 2427H
; 2428H
; 2429H
; 242AH
; 242BH
; 242CH
; 242DH
; 242EH
; 242FH
; 2430H
; 2431H
; 2432H
; 2433H
; 2434H
; 2435H
; 2436H
; 2437H
; 2438H
; 2439H
; 243AH
; 243BH
; 243CH
; 243DH
; 243EH
; 243FH
; 2440H
; 2441H
; 2442H
; 2443H
; 2444H
; 2445H
; 2446H
; 2447H
; 2448H
; 2449H
; 244AH
; 244BH
; 244CH
; 244DH
; 244EH
; 244FH
; 2450H
; 2451H
; 2452H
; 2453H
; 2454H
; 2455H
; 2456H
; 2457H
; 2458H
; 2459H
; 245AH
; 245BH
; 245CH
; 245DH
; 245EH
; 245FH
; 2460H
; 2461H
; 2462H
; 2463H
; 2464H
; 2465H
; 2466H
; 2467H
; 2468H
; 2469H
; 246AH
; 246BH
; 246CH
; 246DH
; 246EH
; 246FH
; 2470H
; 2471H
; 2472H
; 2473H
; 2474H
; 2475H
; 2476H
; 2477H
; 2478H
; 2479H
; 247AH
; 247BH
; 247CH
; 247DH
; 247EH
; 247FH
; 2480H
; 2481H
; 2482H
; 2483H
; 2484H
; 2485H
; 2486H
; 2487H
; 2488H
; 2489H
; 248AH
; 248BH
; 248CH
; 248DH
; 248EH
; 248FH
; 2490H
; 2491H
; 2492H
; 2493H
; 2494H
; 2495H
; 2496H
; 2497H
; 2498H
; 2499H
; 249AH
; 249BH
; 249CH
; 249DH
; 249EH
; 249FH
; 24A0H
; 24A1H
; 24A2H
; 24A3H
; 24A4H
; 24A5H
; 24A6H
; 24A7H
; 24A8H
; 24A9H
; 24AAH
; 24ABH
; 24ACH
; 24ADH
; 24AEH
; 24AFH
; 24B0H
; 24B1H
; 24B2H
; 24B3H
; 24B4H
; 24B5H
; 24B6H
; 24B7H
; 24B8H
; 24B9H
; 24BAH
; 24BBH
; 24BCH
; 24BDH
; 24BEH
; 24BFH
; 24C0H
; 24C1H
; 24C2H
; 24C3H
; 24C4H
; 24C5H
; 24C6H
; 24C7H
; 24C8H
; 24C9H
; 24CAH
; 24CBH
; 24CEH
; 24CFH
; 24D0H
; 24D1H
; 24D2H
; 24D3H
; 24D4H
; 24D5H
; 24D6H
; 24D7H
; 24D8H
; 24D9H
; 24DAH
; 24DBH
; 24DCH
; 24DDH
; 24DEH
; 24DFH
; 24E0H
; 24E1H
; 24E2H
; 24E3H
; 24E4H
; 24E5H
; 24E6H
; 24E7H
; 24E8H
; 24E9H
; 24EAH
; 24EBH
; 24ECH
; 24EDH
; 24EEH
; 24EFH
; 24F0H
; 24F1H
; 24F2H
; 24F3H
; 24F4H
; 24F5H
; 24F6H
; 24F7H
; 24F8H
; 24F9H
; 24FAH
; 24FBH
; 24FCH
; 24FDH
; 24FEH
; 24FFH
; 2500H
; 2501H
; 2502H
; 2503H
; 2504H
; 2505H
; 2506H
; 2507H
; 2508H
; 2509H
; 250AH
; 250BH
; 250CH
; 250DH
; 250EH
; 250FH
; 2510H
; 2511H
; 2512H
; 2513H
; 2514H
; 2515H
; 2516H
; 2517H
; 2518H
; 2519H
; 251AH
; 251BH
; 251CH
; 251DH
; 251EH
; 251FH
; 2520H
; 2521H
; 2522H
; 2523H
; 2524H
; 2525H
; 2526H
; 2527H
; 2528H
; 2529H
; 252AH
; 252BH
; 252CH
; 252DH
; 252EH
; 252FH
; 2530H
; 2531H
; 2532H
; 2533H
; 2534H
; 2535H
; 2536H
; 2537H
; 2538H
; 2539H
; 253AH
; 253BH
; 253CH
; 253DH
; 253EH
; 253FH
; 2540H
; 2541H
; 2542H
; 2543H
; 2544H
; 2545H
; 2546H
; 2547H
; 2548H
; 2549H
; 254AH
; 254BH
; 254CH
; 254DH
; 254EH
; 254FH
; 2550H
; 2551H
; 2552H
; 2553H
; 2554H
; 2555H
; 2556H
; 2557H
; 2558H
; 2559H
; 255AH
; 255BH
; 255CH
; 255DH
; 255EH
; 255FH
; 2560H
; 2561H
; 2562H
; 2563H
; 2564H
; 2565H
; 2566H
; 2567H
; 2568H
; 2569H
; 256AH
; 256BH
; 256CH
; 256DH
; 256EH
; 256FH
; 2570H
; 2571H
; 2572H
; 2573H
; 2574H
; 2575H
; 2576H
; 2577H
; 2578H
; 2579H
; 257AH
; 257BH
; 257CH
; 257DH
; 257EH
; 257FH
; 2580H
; 2581H
; 2582H
; 2583H
; 2584H
; 2585H
; 2586H
; 2587H
; 2588H
; 2589H
; 258AH
; 258BH
; 258CH
; 258DH
; 258EH
; 258FH
; 2590H
; 2591H
; 2592H
; 2593H
; 2594H
; 2595H
; 2596H
; 2597H
; 2598H
; 2599H
; 259AH
; 259BH
; 259CH
; 259DH
; 259EH
; 259FH
; 25A0H
; 25A1H
; 25A2H
; 25A3H
; 25A4H
; 25A5H
; 25A6H
; 25A7H
; 25A8H
; 25A9H
; 25AAH
; 25ABH
; 25ACH
; 25ADH
; 25AEH
; 25AFH
; 25B0H
; 25B1H
; 25B2H
; 25B3H
; 25B4H
; 25B5H
; 25B6H
; 25B7H
; 25B8H
; 25B9H
; 25BAH
; 25BBH
; 25BCH
; 25BDH
; 25BEH
; 25BFH
; 25C0H
; 25C1H
; 25C2H
; 25C3H
; 25C4H
; 25C5H
; 25C6H
; 25C7H
; 25C8H
; 25C9H
; 25CAH
; 25CBH
; 25CEH
; 25CFH
; 25D0H
; 25D1H
; 25D2H
; 25D3H
; 25D4H
; 25D5H
; 25D6H
; 25D7H
; 25D8H
; 25D9H
; 25DAH
; 25DBH
; 25DCH
; 25DDH
; 25DEH
; 25DFH
; 25E0H
; 25E1H
; 25E2H
; 25E3H
; 25E4H
; 25E5H
; 25E6H
; 25E7H
; 25E8H
; 25E9H
; 25EAH
; 25EBH
; 25ECH
; 25EDH
; 25EEH
; 25EFH
; 25F0H
; 25F1H
; 25F2H
; 25F3H
; 25F4H
; 25F5H
; 25F6H
; 25F7H
; 25F8H
; 25F9H
; 25FAH
; 25FBH
; 25FCH
; 25FDH
; 25FEH
; 25FFH
; 2600H
; 2601H
; 2602H
; 2603H
; 2604H
; 2605H
; 2606H
; 2607H
; 2608H
; 2609H
; 260AH
; 260BH
; 260CH
; 260DH
; 260EH
; 260FH
; 2610H
; 2611H
; 2612H
; 2613H
; 2614H
; 2615H
; 2616H
; 2617H
; 2618H
; 2619H
; 261AH
; 261BH
; 261CH
; 261DH
; 261EH
; 261FH
; 2620H
; 2621H
; 2622H
; 2623H
; 2624H
; 2625H
; 2626H
; 2627H
; 2628H
; 2629H
; 262AH
; 262BH
; 262CH
; 262DH
; 262EH
; 262FH
; 2630H
; 2631H
; 2632H
; 2633H
; 2634H
; 2635H
; 2636H
; 2637H
; 2638H
; 2639H
; 263AH
; 263BH
; 263CH
; 263DH
; 263EH
; 263FH
; 2640H
; 2641H
; 2642H
; 2643H
; 2644H
; 2645H
; 2646H
; 2647H
; 2648H
; 2649H
; 264AH
; 264BH
; 264CH
; 264DH
; 264EH
; 264FH
; 2650H
; 2651H
; 2652H
; 2653H
; 2654H
; 2655H
; 2656H
; 2657H
; 2658H
; 2659H
; 265AH
; 265BH
; 265CH
; 265DH
; 265EH
; 265FH
; 2660H
; 2661H
; 2662H
; 2663H
; 2664H
; 2665H
; 2666H
; 2667H
; 2668H
; 2669H
; 266AH
; 266BH
; 266CH
; 266DH
; 266EH
; 266FH
; 2670H
; 2671H
; 2672H
; 2673H
; 2674H
; 2675H
; 2676H
; 2677H
; 2678H
; 2679H
; 267AH
; 267BH
; 267CH
; 267DH
; 267EH
; 267FH
; 2680H
; 2681H
; 2682H
; 2683H
; 2684H
; 2685H
; 2686H
; 2687H
; 2688H
; 2689H

```

```

; This program will provide 3 PWM outputs on HSO pins 0-2
; The input parameters passed to the program are:
;
; HSO_ON_N   HSO on time for pin N
; HSO_OFF_N  HSO off time for pin N
;
; Where: Times are in timer1 cycles
; N takes values from 0 to 3
;
;-----
$INCLUDE(DEN096.INC)
RSEG AT 28H
;
; HSO_ON 0:   DSW 1
; HSO_OFF 0:  DSW 1
; HSO_ON 1:   DSW 1
; HSO_OFF 1:  DSW 1
; OLD_STAT:   ddb 1
; NEW_STAT:   ddb 1
;
;-----
cseg AT 2080H
LD SP,$100H
LD HSO_ON_0,$100H ; Set initial values
LD HSO_OFF_0,$400H ; Note that times must be long enough
LD HSO_ON_1,$280H ; to allow the routine to run after each
LD HSO_OFF_1,$280H ; line change.
ANDB OLD_STAT,$050,$0FH
XORB OLD_STAT,$0FH

wait: JBS IOS0,6,wait ; Loop until HSO holding register
NOP ; is empty

; For operation with interrupts 'store_stat:' would be the
; entry point of the routine.
; Note that a DI or PUSHF might have to be added.

store_stat:
ANDB NEW_STAT,$050,$0FH ; Store new status of HSO
CMPB OLD_STAT,NEW_STAT
JE wait ; If status hasn't changed
XORB OLD_STAT,NEW_STAT

check_0:
JBC OLD_STAT,0,check_1 ; Jump if OLD_STAT(0)=NEW_STAT(0)
JBS NEW_STAT,0,set_off_0

set_on_0:
LDB HSO_COMMAND,$00110000B ; Set HSO for timer1, set pin 0
ADD HSO_TIME,TIMER1,HSO_OFF_0 ; Time to set pin = Timer1 value
BR check_1 ; + Time for pin to be low

set_off_0:
LDB HSO_COMMAND,$00010000B ; Set HSO for timer1, clear pin 0
ADD HSO_TIME,TIMER1,HSO_ON_0 ; Time to clear pin = Timer1 value
BR check_1 ; + Time for pin to be high

check_1:
JBC OLD_STAT,1,check_done ; Jump if OLD_STAT(1)=NEW_STAT(1)
JBS NEW_STAT,1,set_off_1

set_on_1:
LDB HSO_COMMAND,$00110001B ; Set HSO for timer1, set pin 1
ADD HSO_TIME,TIMER1,HSO_OFF_1 ; Time to set pin = Timer1 value
BR check_done ; + Time for pin to be low

set_off_1:
LDB HSO_COMMAND,$00010001B ; Set HSO for timer1, clear pin 1
ADD HSO_TIME,TIMER1,HSO_ON_1 ; Time to clear pin = Timer1 value
BR check_done ; + Time for pin to be high

check_done:
LDB OLD_STAT,NEW_STAT ; Store current status and
; wait for interrupt flag

BR wait

END

```

270061-25

Listing 3-8. Generating a PWM with the HSO

laration section (i.e., the section between "RSEG" and "CSEG") in Listing 3-8 must be changed to that in Listing 3-9.

The driver routine simply changes the duty cycle of the waveform and sets the second HSO output to a fre-

quency twice that of the first one. A slightly different driver routine could easily be the basis for a switching power supply or a variable frequency/variable voltage motor driver. The listing of the driver routine is shown in Listing 3-10.

```
; NOTE: Use this file to replace the declaration section of
; the HSO PWM program from "$INCLUDE(DEMO96.INC)" through
; the line prior to the label "wait". Also change the last
; branch in the program to a "RET".
RSEG
```

```
Listing 3-10. Driver Module for HSO PWM Program (Continued)
DSTAT: DSB 1
extrn HSO_ON_0:word, HSO_OFF_0:word
extrn HSO_ON_1:word, HSO_OFF_1:word
extrn HSO_TIME:word, HSO_COMMAND:byte
extrn TIMER1:word, IOS0:byte
extrn SP:word

public OLD_STAT
old_stat: ddb 1
new_stat: ddb 1

cseg
PUBLIC wait
wait:
    ; ... (rest of the code) ...
    RET
```

270061-26

Listing 3-9. Changes to Declarations for HSO Routine

```
$TITLE('HSODRV.APT: Driver module for HSO PWM program')
HSODRV MODULE MAIN, STACKSIZE(8)

PUBLIC HSO_ON_0, HSO_OFF_0
PUBLIC HSO_ON_1, HSO_OFF_1
PUBLIC HSO_TIME, HSO_COMMAND
PUBLIC SP, TIMER1, IOS0
```

```
$INCLUDE(DEMO96.INC)
```

```
rseg at 28H
```

```
EXTRN OLD_STAT:byte
HSO_ON_0: ddb 1
HSO_OFF_0: ddb 1
HSO_ON_1: ddb 1
HSO_OFF_1: ddb 1
count: ddb 1
```

```
cseg at 2080H
```

```
EXTRN wait:entry
```

```
strt: DI
LD SP, #1000H
ANDB OLD_STAT, IOS0, #0FH
XORB OLD_STAT, #0FH
```

```
initial:
```

```
LD CX, #0100H
```

```
loop:
```

```
LD AX, #1000H
SUB BX, AX, CX
LD AX, CX
```

```
ST AX, HSO_ON_0
BX, HSO_OFF_0
```

270061-27

Listing 3-10. Driver Module for HSO PWM Program

that the HSO unit can be used to initiate the desired tasks at the appropriate tooth count. The interrupt routine initiated by HSI.0 can be used to perform any software task required every revolution. In this system, the overhead which would normally require extensive software has been done with the hardware on the 8096, thus making more software time available for control programs.

3.2.3. USING THE SERIAL PORT IN MODE 1

Mode 1 of the serial port supports the basic asynchronous 8-bit protocol and is used to interface to most CRTs and printers. The example in Listing 3-11 shows a simple routine which receives a character and then

transmits the same character. The code is set up so that minor modifications could make it run on an interrupt basis. Note that it is necessary to set up some flags a initial conditions to get the routine to run properly. If it was desired to send 7 bits of data plus parity instead of 8 bits of data the PEN bit would be set to a one. Inter-processor communication, as described in section 2.3.4, can be set up by simply adding code to change RB8 and the port mode to the listing below. The hardware shown in Figure 3-2 can be used to convert the logic level output of the 8096 to ± 12 or 15 volt levels to connect to a CRT. This circuit has been found to work with most RS-232 devices, although it does not conform to strict RS-232 specifications. If true RS-232 conformance is required then any standard RS-232 driver can be used.

```

$TITLE('SP.APT: SERIAL PORT DEMO PROGRAM')

$INCLUDE(DEMO96.INC)

rseg      at 28H

        CHR:   ddb      1
        SPTMP: ddb      1
        TEMP0: ddb      1
        TEMP1: ddb      1
        RCV_FLAG: ddb      1

cseg      at 200CH

        DCW      ser_port_int

cseg      at 2080H

        LD      SP, #100H

        LDB      IOCL, #00100000B      ; Set P2.0 to TXD

        ; Baud rate = input frequency / (64*baud_val)
        ; baud_val = (input frequency/64) / baud rate

        baud_val equ      39      ; 39 = (12,000,000/64)/4800 baud

        BAUD_HIGH equ      ((baud_val-1)/256) OR 80H      ; Set MSB to 1
        BAUD_LOW  equ      (baud_val-1) MOD 256

        LDB      BAUD_REG, #BAUD_LOW
        LDB      BAUD_REG, #BAUD_HIGH

        LDB      SPCON, #01001001B      ; Enable receiver, Mode 1

        ; The serial port is now initialized

        STB      SBUF, CHR      ; Clear serial Port
        LDB      TEMP0, #00100000B      ; Set TI-temp

        LDB      INT_MASK, #01000000B      ; Enable Serial Port Interrupt
        EI

loop:     BR      loop      ; Wait for serial port interrupt

ser_port_int:
        PUSHF

rd_again:

        LDB      SPTMP, SPSTAT      ; This section of code can be replaced
        ORB      TEMP0, SPTMP      ; with "ORB TEMP0, SP_STAT" when the
        ANDB      SPTMP, #01100000B      ; serial port TI and RI bugs are fixed
        JNE      rd_again      ; Repeat until TI and RI are properly cleared

```

Listing 3-11. Using the Serial Port in Mode 1

```

get_byte:
JBC     TEMP0, 6, put_byte
STB     SBUP, CHR
ANDB    TEMP0, #10111111B
LDB     RCV_FLAG, #OFFH

put_byte:
JBC     RCV_FLAG, 0, continue
JBC     TEMP0, 5, continue
LDB     SBUP, CHR
ANDB    TEMP0, #11011111B

ANDB    CHR, #01111111B
CMPB    CHR, #00DH
JNE     clr_rcv
LDB     CHR, #00AH
BR      continue

clr_rcv:
CLRB    RCV_FLAG

continue:
POPF
RET

END

```

; If RI-temp is not set
; Store byte
; CLR RI-temp
; Set bit-received flag

; If receive flag is cleared
; If TI was not set
; Send byte
; CLR TI-temp

; This section of code appends
; an LF after a CR is sent

; Clear bit-received flag

270061-31

Listing 3-11. Using the Serial Port in Mode 1 (Continued)

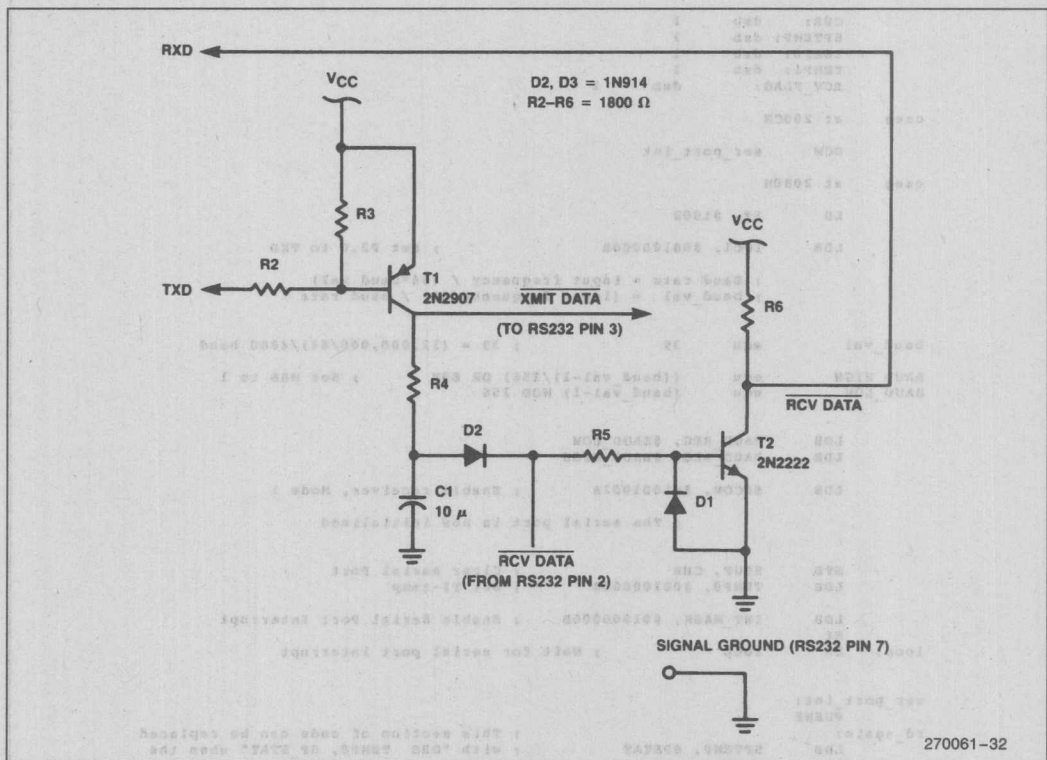


Figure 3-2. Serial Port Level Conversion

3.2.4. USING THE A TO D

The code in Listing 3-12 makes use of the software flags to implement a non-interrupt driven routine which scans A to D channels 0 through 3 and stores them as words in RAM. An interrupt driven routine is shown in section 4.1. When using the A to D it is important to always read the value using the byte read commands, and to give the converter 8 state times to start converting before reading the status bit.

Since there is no sample and hold on the A to D converter it may be desirable to use an RC filter on each input. A 100Ω resistor in series with a 0.22 μF capacitor to ground has been used successfully in the lab. This circuit gives a time constant of around 22 microseconds which should be long enough to get rid of most noise, without overly slowing the A to D response time.

4.0 ADVANCED SOFTWARE EXAMPLES

Using the 8096 for applications which consist only of the brief examples in the previous section does not

really make use of its full capabilities. The following examples use some of the code blocks from the previous section to show how several I/O features can be used together to accomplish a practical task. Three examples will be shown. The first is simply a combination of several of the section 3 examples run under an interrupt system. Next, a software serial port using the HSIO unit is described. The concluding example is one of interfacing the HSI unit to an optical encoder to control a motor.

4.1. Simultaneous I/O Routines under Interrupt Control

A four channel analog to PWM converter can easily be made using the 8096. In the example in Listing 4 analog channels are read and 3 PWM waveforms are generated on the HSO lines and one on the PWM pin. Each analog channel is used to set the duty cycle of its associated output pin. The interrupt system keeps the whole program humming, providing time for a background task which is simply a 32 bit software counter. To show which routines are executing and in which

```

$TITLE('ATOD.APT: SCANNING THE A TO D CHANNELS')
$INCLUDE(DEMO96.INC)

RSEG    at 28H

        BL      EQU      BX:BYTE
        DL      EQU      DX:BYTE

RESULT_TABLE:
        RESULT_1:      dsw      1
        RESULT_2:      dsw      1
        RESULT_3:      dsw      1
        RESULT_4:      dsw      1

cseg    at 2080H

start:  LD      SP, #100H      ; Set Stack Pointer
        CLR     BX

next:   ADDB     AD_COMMAND,BL, #1000B      ; Start conversion on channel
                                           ; indicated by BL register

        NOP     ; Wait for conversion to start

check:  JBS     AD_RESULT_LO, 3, check      ; Wait while A to D is busy

        LDB     AL, AD_RESULT_LO      ; Load low order result
        LDB     AH, AD_RESULT_HI      ; Load high order result

        ADDB     DL, BL, BL      ; DL=BL*2
        LDB     DX, DL
        ST       AX, RESULT_TABLE[DX]      ; Store result indexed by BL*2

        INCB     BL      ; Increment BL modulo 4
        ANDB     BL, #03H

        BR      next

END

```

Listing 3-12. Scanning the A to D Channels

270061-33

order, Port 1 output pins are used to indicate the current status of each task. The actual code listing is included in Appendix B. The initialization section, shown in Listing 4-1a, clears a few variables and then loads the first set of on and off times to the HSO unit. Note that 8 state times must

be waited between consecutive loads of the HSO. If this is not done it is possible to overwrite the contents of the CAM holding register. An A/D interrupt is forced by setting the bit in the Interrupt Pending register. This causes the first A/D interrupt to occur just after the Interrupt Mask register is set and interrupts are enabled.

Listing 4-1. Using Multiple I/O Devices

```

$TITLE ('8096 EXAMPLE PROGRAM FOR PWM OUTPUTS FROM A TO D INPUTS')
$PAGEWIDTH(130)

; This program will provide 3 PWM outputs on HSO pins 0-2
; and one on the PWM.
; The PWM values are determined by the input to the A/D converter.
;
$INCLUDE(Demo96.inc)

RSEG AT 28H
DL EQU DX:BYTE

ON_TIME:
    PWM_TIME_1: DSW 1
    HSO_ON_0: DSW 1
    HSO_ON_1: DSW 1
    HSO_ON_2: DSW 1

RESULT_TABLE:
    RESULT_0: DSW 1
    RESULT_1: DSW 1
    RESULT_2: DSW 1
    RESULT_3: DSW 1

    NXT_ON_T: DSW 1
    NXT_OFF_0: DSW 1
    NXT_OFF_1: DSW 1
    NXT_OFF_2: DSW 1
    COUNT: DSW 1
    AD_NUM: DSW 1
    TMP: DSW 1
    HSO_PER: DSW 1
    LAST_LOAD: DSW 1

; Channel being converted

cseg AT 2000H
    DCW start ; Timer_ovf_int
    DCW Atod_done_int
    DCW start ; HSI_data_int
    DCW HSO_exec_int

cseg AT 2080H
    start: LD SP, #100H ; Set Stack Pointer
    CLR AX
    wait: DEC AX ; wait approx. 0.2 seconds for
    JNE wait ; SBE to finish communications

    CLR AD_NUM
    LD PWM_TIME_1, #080H
    LD HSO_PER, #100H
    LD HSO_ON_0, #040H
    LD HSO_ON_1, #080H
    LD HSO_ON_2, #0C0H
    ADD NXT_ON_T, Timer1, #100H

```

270061-34

Listing 4-1a. Initializing the A to D to PWM Program

```

LDB HSO_COMMAND, #00110110B ; Set HSO for timer1, set pin 0,1
LD HSO_TIME, NXT_ON_T ; with interrupt
NOP
NOP
LDB HSO_COMMAND, #00100010B ; Set HSO for timer1, set pin 2
ADD HSO_TIME, NXT_ON_T ; without interrupt

ORB LAST_LOAD, #00000111B ; Last loaded value was set all pins
LDB INT_MASK, #00001010B ; Enable HSO and A/D interrupts
LDB INT_PENDING, #00001010B ; Fake an A/D and HSO interrupt
EI

loop: ORB Port1, #00000001B ; set P1.0
      ADD COUNT, #01
      ADDC COUNT+2, zero
      ANDB Port1, #11111110B ; clear P1.0
      BR loop

```

270061-35

Listing 4-1a. Initializing the A to D to PWM program (Continued)

```

HSD_EXECUTED_INTERRUPT
HSD_exec_int:
PUSHF
ORB Port1, #00000010B ; Set p1.1

SUB TMP, TIMER1, NXT_ON_T
CMP TMP, ZERO
JLT set_off_times

set_on_times:
ADD NXT_ON_T, HSD_PER
LDB HSO_COMMAND, #00110110B ; Set HSO for timer1, set pin 0,1
LD HSO_TIME, NXT_ON_T
NOP
LDB HSO_COMMAND, #00100010B ; Set HSO for timer1, set pin 2
LD HSO_TIME, NXT_ON_T
ORB LAST_LOAD, #00000111B ; Last loaded value was all ones
LDB PWM_CONTROL, PWM_TIME_1 ; Now is as good a time as any
BR check_done ; to update the PWM reg

set_off_times:
JBC LAST_LOAD, 0, check_done

ADD NXT_OFF_0, NXT_ON_T, HSD_ON_0
LDB HSO_COMMAND, #00010000B ; Set HSO for timer1, clear pin 0
LD HSO_TIME, NXT_OFF_0
NOP
ADD NXT_OFF_1, NXT_ON_T, HSD_ON_1
LDB HSO_COMMAND, #00010001B ; Set HSO for timer1, clear pin 1
LD HSO_TIME, NXT_OFF_1
NOP
ADD NXT_OFF_2, NXT_ON_T, HSD_ON_2
LDB HSO_COMMAND, #00010010B ; Set HSO for timer1, clear pin 2
LD HSO_TIME, NXT_OFF_2
ANDB LAST_LOAD, #11111000B ; Last loaded value was all 0s

check_done:
ANDB Port1, #11111110B ; Clear P1.1
POPF
RET

```

270061-36

Listing 4-1b. Interrupt Driven HSD Routine

```

; A to D COMPLETE INTERRUPT
ATOD_done_int:
    PUSHF
    ORB     Port1, #00000100B      ; Set Pl.2
    ANDB    AL, AD_RESULT_LO, #11000000B ; Load low order result
    LDB     AH, AD_RESULT_HI      ; Load high order result
    ADDB    DL, AD_NUM, AD_NUM    ; DL= AD_NUM *2
    LDB     DX, DL
    ST      AX, RESULT_TABLE[DX]  ; Store result indexed by DX
    CMPB    AL, #01000000B        ; Round up if needed
    JNB     no_rnd
    CMPB    AH, #0FFH             ; Don't increment if AH=0FFH
    JE      no_rnd
    INCB    AH
no_rnd:
    LDB     AL, AH                ; Align byte and change to word
    CLRB    AH
    ST      AX, ON_TIME[DX]
    INCB    AD_NUM
    ANDB    AD_NUM, #003H        ; Keep AD_NUM between 0 and 3
next:
    ADDB    AD_COMMAND, AD_NUM, #1000B ; Start conversion on channel
    ; indicated by AD_NUM register
    ANDB    Port1, #11111011B    ; Clear Pl.2
    POPF
    RET
END

```

270061-37

Listing 4-1c. Interrupt Driven A to D Routine

The HSO routine shown in Listing 4-1b is slightly different than the one in section 3. All of the HSO lines turn on at the same time, only the turn-off-time is varied between lines. This action is what is most commonly required for multiple PWM outputs and simplifies the software. A comparison is made between Timer1 and the next HSO turn on time at the beginning of the routine. If the next turn on time has passed, then the on-times are loaded into the CAM, otherwise the off times are loaded.

The maximum number of events in the CAM at any given time is 7. This occurs when the first line to turn off does so, causing the off-times for all of the lines to be loaded. For two of the lines there will be an offtime, an on-time, and the just loaded off-time. The other line (the one that just turned off) will have only the on-time and the just loaded off-time.

A/D conversions are performed by the code in Listing 4-1c about every 60 microseconds, 42 for the conversion, the rest for overhead. The A/D routine sets up the HSO and PWM on and off times. Since the A/D

has a ten bit output, the most significant 8 bits are rounded up or down based on the least significant two bits.

4.2. Software Serial Port Using the HSIO Unit

There are many systems which require more than one serial port, an example is a system which must communicate with other computers and have an additional port for a local console. If the on-board UART is being used as an inter-processor link, the HSIO unit can be used to interface the 8096 to an additional asynchronous line.

Figure 4-1 shows the format of a standard 10-bit asynchronous frame. The start bit is used to synchronize the receiver to the transmitter; at the leading edge of the START bit the receiver must set up its timing logic to sample the incoming line in the center of each bit. Following the start bit are the eight data bits which are transmitted least significant bit first. The STOP bit is set to the opposite state of the START bit to guar-

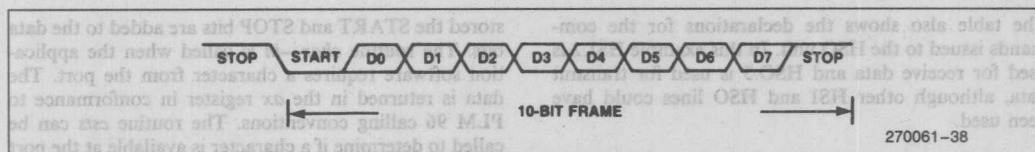


Figure 4-1. 10-bit Asynchronous Frame

antee that the leading edge of the START bit will cause a transition on the line; it also provides for a dead time on the line so that the receiver can maintain its synchronization.

The remainder of this section will show how a full-duplex asynchronous port can be built from the HSO unit. There are four sections to this code:

1. Interface routines. These routines provide a procedural interface between the interrupt driven core of the software serial port and the remainder of the application software.
2. Initialization routine. This routine is called during the initialization of the overall system and sets up the various variables used by the software port.

3. Transmit ISR. This routine runs as an ISR (interrupt service routine) in response to an HSO interrupt interrupt. Its function is to serialize the data passed to it by the interface routines.

4. Receive ISRs. There are two ISRs involved in the receive process. One of them runs in response to an HSI interrupt and is used to synchronize the receive process at the leading edge of the start bit. The second receive ISR runs in response to an HSO generated software timer interrupt, this routine is scheduled to run at the center of each bit and is used to deserialize the incoming data.

The routines share the set of variables that are shown in Listing 4-2. These variables should be accessed only by the routines which make up the software serial port.

VARIABLES NEEDED BY THE SOFTWARE SERIAL PORT			

rseg			
rcve_state:	dab 1		
rxrdy	equ 1		
rxoverrun	equ 2		
rip	equ 4		
rcve_buf:	dab 1		
rcve_reg:	dab 1		
sample_time:	daw 1		
serial_out:	daw 1		
baud_count:	daw 1		
txd_time:	daw 1		
char:	dab 1		
COMMANDS ISSUED TO THE HSO UNIT			

mark_command	equ 0110101b		
space_command	equ 0010101b		
sample_command	equ 0011000b		
\$reject			

Listing 4-2. Software Serial Port Declarations

mark_command	equ 0110101b		
space_command	equ 0010101b		
sample_command	equ 0011000b		
\$reject			

The table also shows the declarations for the commands issued to the HSO unit. In this example HSI.2 is used for receive data and HSO.5 is used for transmit data, although other HSI and HSO lines could have been used.

The interface routines are shown in Listing 4-3. Data is passed to the port by pushing the eight-bit character into the stack and calling *char_out*, which waits for any in-process transmission to complete and stores the character into the variable *serial_out*. As the data is

stored the START and STOP bits are added to the data bits. The routine *char_in* is called when the application software requires a character from the port. The data is returned in the *ax* register in conformance to PLM 96 calling conventions. The routine *csts* can be called to determine if a character is available at the port before calling *char_in*. (If no character is available *char_in* will wait indefinitely).

The initialization routine is shown in Listing 4-4. This routine is called with the required baud rate in the

```

; char_out:
; Output character to the software serial port
;
; pop cx ; the return address
; pop bx ; the character for output
; ldb (bx+1),%01h ; add the start and stop bits
; add bx,bx ; to the char and leave as 16 bit
; wait_for_xmit:
; cmp serial_out,0 ; wait for serial out=0 (it will be cleared by
; bne wait_for_xmit ; the hso interrupt process)
; st bx,serial_out ; put the formatted character in serial_out
; br [cx] ; return to caller

; csts:
; Returns "true" (ax<>0) if char_in has a character.
;
; clr ax
; bbc rcve_state,0,csts_exit
; inc ax
; csts_exit:
; ret

; char_in:
; Get a character from the software serial port
;
; ; wait for character ready
; bbc rcve_state,0,char_in
; pushf ; set up a critical region
; andb rcve_state,%not(rxdy)
; ldbz al,rcve_buf
; popf ; leave the critical region
; ret

```

270061-40

Listing 4-3. Software Serial Port Interface Routines

```

; setup_serial_port:
; Called on system reset to initiate the software serial port.
;
; pop cx ; the return address
; pop bx ; the baud rate (in decimal)
; ld dx,%00007h ; dx:ax:=500,000 (assumes 12 Mhz crystal)
; ld ax,%0A120h
; divu ax,bx ; calculate the baud count (500,000/baudrate)
; st ax,baud_count
; st 0,serial_out ; clear serial out
; ldb loc1,%01100000b ; Enable HSO.5 and Txd
; bbs loc0,6,$ ; Wait for room in the HSO CAM
; ; and issue a MARK command.
; add txd_time,timer1,20
; ldb hso_command,%mark_command
; ld hso_time,txd_time
; clrb rcve_buf ; clear out the receive variables
; clrb rcve_reg
; clrb rcve_state
; call init_receive ; setup to detect a start bit
; br [cx] ; return

```

270061-41

Listing 4-4. Software Serial Port Initialization Routine

stack; it calculates the bit time from the baud rate and stores it in the variable *baud_count* in units of TIMER1 ticks. An HSO command is issued which will initiate the transmit process and then the remainder of the variables owned by the port are initialized. The routine *init_receive* is called to setup the HSI unit to look for the leading edge of the START bit.

The transmit process is shown in Listing 4-5. The HSO unit is used to generate an output command to the transmit pin once per bit time. If the *serial_out* register is zero a MARK (idle condition) is output. If the *serial_out* register contains data then the least sig-

nificant bit is output and the register shifted right one place. The framing information (START and STOP bits) are appended to the actual data by the interface routines. Note that this routine will be executed once per bit time whether or not data is being transmitted. It would be possible to use this routine for additional low resolution timing functions with minimal overhead.

The receive process consists of an initialization routine and two interrupt service routines, *hsi_isr* and *software_timer_isr*. The listings of these routines are shown in Listings 4-6a, 4-6b, and 4-6c respectively. The

```

;
; hso_isr:
; Fields the hso interrupts and performs the serialization of the data.
; Note: this routine would be incorporated into the hso service strategy for an
; actual system.
;
;-----
; at 2006h
; hso_isr ; Set up vector
;-----

;-----
; tx time, baud_count
; serial_out, 0 ; if character is done send a mark
; send_mark
; serial_out, #1 ; else send bit 0 of serial_out and shift
; send_mark ; serial_out left one place.
;
; send_space:
; ldb hso_command, $space_command
; ld hso_time, tx time
; br hso_isr_exit
;
; send_mark:
; ldb hso_command, $mark_command
; ld hso_time, tx time
;
; hso_isr_exit:
; popf
; ret
;
; $eject

```

Listing 4-5. Software Serial Port Transmit Process

Listing 4-6. Receive Process

```

;
; init_receive:
; Called to prepare the serial input process to find the leading edge of
; a start bit.
;
;
; ldb loc0, $00000000b ; disconnect change detector
; ldb hsi_mode, $00100000b ; negative edges on HSI.2
;
; flush_fifo:
; orl iosl_save, iosl
; bbl iosl_save, 7, flush_fifo_done
; ldb al, hsi_status
; ld ax, hsi_time ; trash the fifo entry
; andb iosl_save, $not(80h) ; clear bit 7.
; br flush_fifo
;
; flush_fifo_done:
; ldb loc0, $00010000b ; connect HSI.2 to detector
; ret

```

Listing 4-6a. Software Serial Port Receive Initialization

```

; hsi_isr: interrupts from the HSI unit, used to detect the leading edge
; of the START bit
; Note: this routine would be incorporated into the HSI strategy of an actual
; system.
cseg at 2004h
dcw hsi_isr ; setup the interrupt vector

cseg
pushf
push ax
ldb al, hsi_status
ld sample_time, hsi_time
bbc al, 4, exit_hsi
bbs ios0, 7, $
ld ax, baud_count
shr ax, #1
add sample_time, ax
ldb hso_command, #sample_command
st sample_time, hso_time
ldb loc0, #00000000b ; disconnect hsi.2 from change detector

exit_hsi:
pop
popf
ret

```

Listing 4-6b. Software Serial Port Start Bit Detect

```

; software_timer_isr:
; Fields the software timer interrupt, used to deserialize the incoming data.
; Note: this routine would be incorporated into the software timer strategy
; in an actual system.
cseg at 200ah
dcw software_timer_isr ; setup vector

cseg
pushf
orb iosl_save, iosl
andb iosl_save, #not(01h) ; clear bit 0
andb 0, rcve_state, #0fch ; All bits except rxrdy and overrun=0
bne process_data

process_start_bit:
bbc hsi_status, 5, start_ok
call init_receive
software_timer_exit

start_ok:
orb rcve_state, #rip ; set receive in progress flag
br schedule_sample

process_data:
bbs rcve_state, 7, check_stopbit
shrb rcve_reg, #1
bbc hsi_status, 5, datazero
orb rcve_reg, #80h ; set the new data bit

datazero:
addb rcve_state, #10h ; increment bit count
br schedule_sample

check_stopbit:
bbc hsi_status, 5, $ ; DEBUG ONLY
ldb rcve_buf, rcve_reg
orb rcve_state, #rxrdy
andb rcve_state, #03h ; Clear all but ready and overrun bits
call init_receive
software_timer_exit

schedule_sample:
bbs ios0, 7, $ ; wait for holding reg empty
ldb hso_command, #sample_command
add sample_time, baud_count
st sample_time, hso_time

software_timer_exit:
popf
ret

```

Listing 4-6c. Software Serial Port Data Reception

ware timer interrupt in one-half of a bit time. This first sample is used to verify that the START bit has not ended prematurely (a protection against a noisy line). The software timer service routine uses the variable `rcv_state` to determine whether it should check for a valid START bit. When a complete character has been received it is moved to the receive buffer and `init_receive` is called to set up the receive process for the next character. This routine is also called when an error (e.g., invalid START bit) is detected.

Appendix C contains the complete listing of the routines and the simple loop which was used to initialize them and verify their operation. The test was run for several hours at 9600 baud with no apparent malfunction of the port.

4.3. Interfacing an Optical Encoder to the HSI Unit

Optical encoders are among one of the more popular devices used to determine position of rotating equipment. These devices output two pulse trains with edges that occur from 2 to 4000 times a revolution.

requently there is a third line which generates one pulse per revolution for indexing purposes. Figure 4-2 shows a six line encoder and typical waveforms. As can be seen, the two waveforms provide the ability to determine both position and direction. Since a microcontroller can perform real time calculations it is possible to determine velocity and acceleration from the position and time information.

Interfacing to the encoder can be an interesting problem, as it requires connecting mechanically generated electrical signals to the HSI unit. The problems arise because it is difficult to obtain the exact nature of the signals under all conditions.

The equipment used in the lab was a Pittman 9400 series gearmotor with a 600 line optical encoder from Vernitech. The encoder has to be carefully attached to the shaft to minimize any runout or endplay. Fortunately, Pitmann has started marketing their motors with ball bearings and optical encoders already installed. It is recommended that the encoder be mounted to the motor using the exact specifications of the encoder manufacturer and/or a good machine shop.

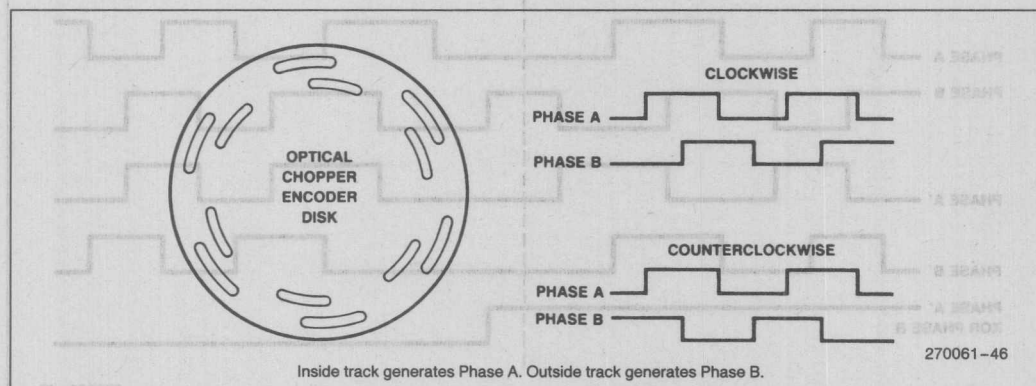


Figure 4-2. Optical Encoder and Waveforms

intel

encoder and after the digital filter are shown in Figure 4-3. The circuitry connecting the encoder to the 8096 requires only two chips. A one-shot constructed of XOR gates generates pulses on each edge of each signal. The pulses generated by Phase A are used to clock the signal from Phase B and vice versa. The hardware is shown in Figure 4-4. CMOS parts are used to reduce loading on the encoder so that buffers are not needed. Note that T2CLK is clocked on both edges of both filtered phases.

By using this method repetitive edges on a single phase without an edge on the other phase will not be passed on to the 8096. Repetitive edges on a phase can occur when the motor is stopped and vibrates or when it is changing direction. The digital filtering technique causes a little more delay in the signal at slow speeds than an analog filter would, but the simplicity trade off is worthwhile. The net effect of digital filtering is losing the ability to determine the first edge after a direction change. This does not affect the count since the first edge in both directions is lost.

directly to the 8096. As these would be input signals, Port 0 is the most likely choice for connection. It would not be required to connect these lines to the HSI unit, as the information on them would only be needed when the motor is going very slowly.

The motor is driven using the PWM output pin for power control and a port pin for direction control. The 8096 drives a 7438 which drives 2 opto-isolators. These in turn drive two VFETs. A MOV (Metal Oxide Varistor, a type of transient absorber) is used to protect the VFETs, and a capacitor filters the PWM to get the best motor performance. Figure 4-5 shows the driver circuitry. To avoid noise getting into the 8096 system, the ± 15 volt power supply is isolated from the 8096 logic power supply.

This is the extent of the external circuitry required for this example. All of the counting and direction detection are done by the 8096. There are two sections to the example: driving the motor and interfacing to the encoder. The motor driver uses proportional control with

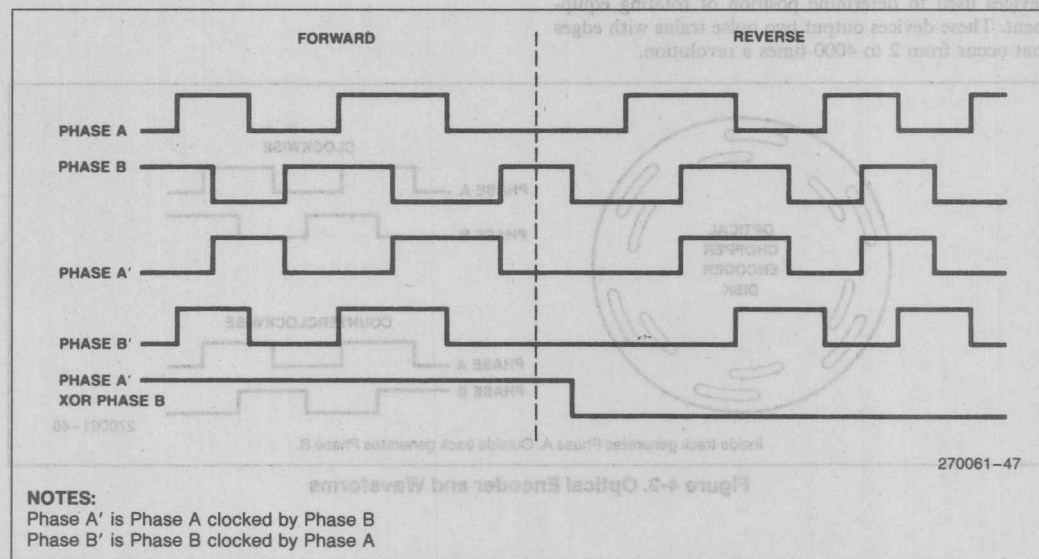


Figure 4-3. Filtered Encoder Waveforms

some modifications and a braking algorithm. Since the main point of this example is I/O interfacing, the motor driver will be briefly described at the end of this section.

In order to interface to the encoder it is necessary to know the types of waveforms that can be expected. The motor was accelerated and decelerated many times using different maximum voltages. It was found that the

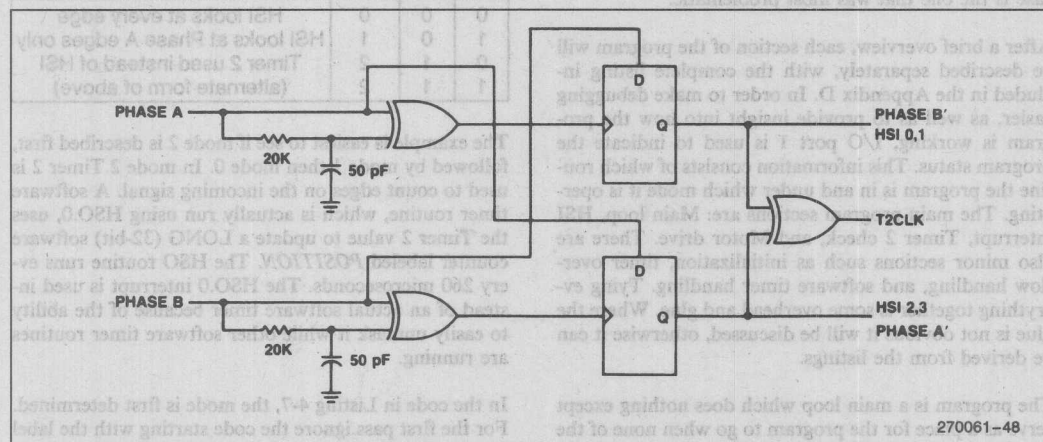


Figure 4-4. Schematic of Optical Encoder to 8096 Interface

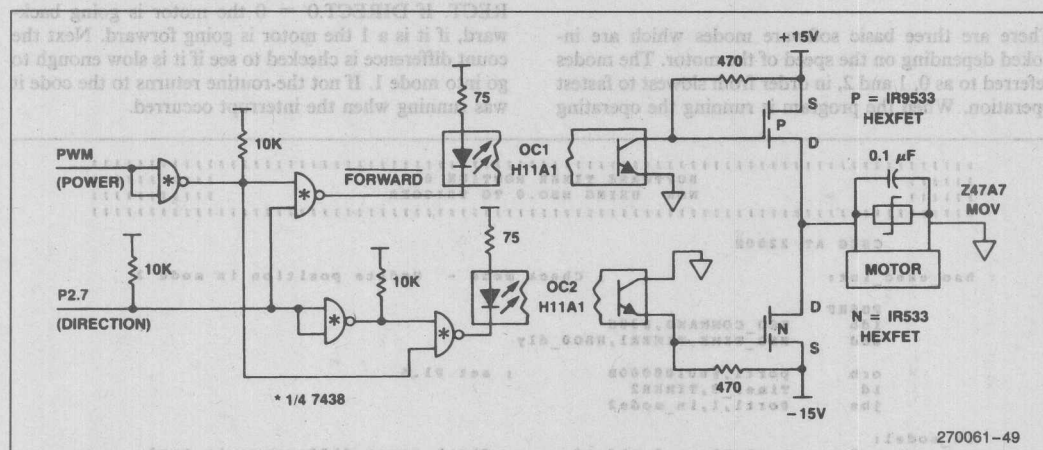


Figure 4-5. Motor Driver Circuitry

motor would decelerate smoothly until the time between encoder edges was around 100 microseconds. At this point the motor would either continue to decelerate slowly, or would suddenly stop and reverse. The latter case is the one that was most problematic.

After a brief overview, each section of the program will be described separately, with the complete listing included in the Appendix D. In order to make debugging easier, as well as to provide insight into how the program is working, I/O port 1 is used to indicate the program status. This information consists of which routine the program is in and under which mode it is operating. The main program sections are: Main loop, HSI interrupt, Timer 2 check, and Motor drive. There are also minor sections such as initialization, timer overflow handling, and software timer handling. Tying everything together is some overhead and glue. Where the glue is not obvious it will be discussed, otherwise it can be derived from the listings.

The program is a main loop which does nothing except serve as a place for the program to go when none of the interrupt routines are being run. All of the processing is done on an interrupt basis.

There are three basic software modes which are invoked depending on the speed of the motor. The modes referred to as 0, 1 and 2, in order from slowest to fastest operation. When the program is running the operating

mode is indicated by the lower 2 bits of Port 1, with the following coding:

P1.0	P1.1	Mode	Description
0	0	0	HSI looks at every edge
1	0	1	HSI looks at Phase A edges only
0	1	2	Timer 2 used instead of HSI
1	1	2	(alternate form of above)

The example is easiest to see if mode 2 is described first, followed by mode 1 then mode 0. In mode 2 Timer 2 is used to count edges on the incoming signal. A software timer routine, which is actually run using HSO.0, uses the Timer 2 value to update a LONG (32-bit) software counter labeled *POSITION*. The HSO routine runs every 260 microseconds. The HSO.0 interrupt is used instead of an actual software timer because of the ability to easily unmask it while other software timer routines are running.

In the code in Listing 4-7, the mode is first determined. For the first pass ignore the code starting with the label *in_mode_1*. Starting with *in_mode_2* the counter is incremented or decremented based on bit zero of *DIRECT*. If *DIRECT.0* = 0 the motor is going backward, if it is a 1 the motor is going forward. Next the count difference is checked to see if it is slow enough to go into mode 1. If not the routine returns to the code it was running when the interrupt occurred.

```

SOFTWARE TIMER ROUTINE 0
NOW USING HSO.0 TO TRIGGER

CSEG AT 2280H
hso_exec_int:
    PUSHF
    ldb     HSO_COMMAND,$30H
    add     HSO_TIME,TIMER1,HSO0_dly

    orb     port1,$00100000B    ; set P1.5
    ld      timer_2,TIMER2
    jbs     port1.1,in_mode2

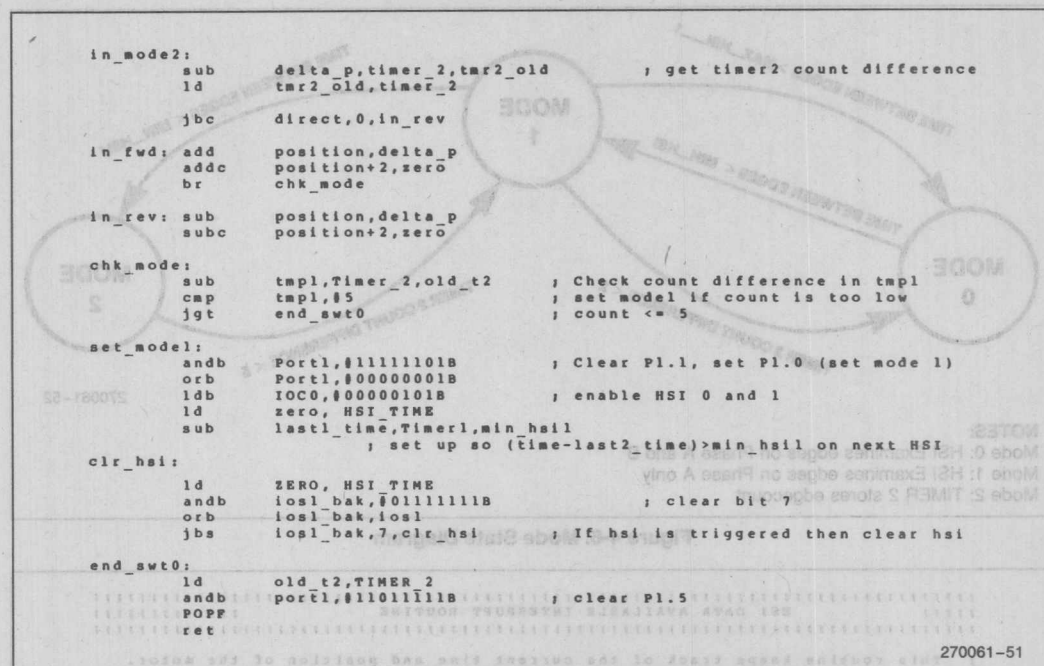
in_mode1:
    sub     tmp1,timer_2,old_t2    ; Check count difference in tmp1
    cmp     tmp1,$2
    jh      end_swt0

set_mode0:
    jbc     port1.0,end_swt0        ; if already in mode 0
    andb    port1,$11111100B      ; Clear P1.0, P1.1 (set mode 0)
    ldb     IOC0,$01010101B      ; enable all HSI
    ldb     last_stat,zero
    br      end_swt0

```

270061-50

Listing 4-7. Motor Control HSO.0 Timer Routine



Listing 4-7. Motor Control HSO.0 Timer Routine (Continued)

If the pulse rate is slow enough to go to mode 1, the transition is made by enabling HSI.0 and HSI.1. Both of these lines are connected to the same encoder line, with HSI.0 looking for rising edges and HSI.1 looking for falling edges. The *HSI_TIME* register is read to speed up clearing the HSI FIFO and the *LAST1_TIME* value is set up so the mode 1 routine does not immediately put the program into another mode. The HSI FIFO is then cleared, the Timer 2 value used throughout this routine is saved, and the routine returns.

This routine still runs in modes 0 and 1, but in an abbreviated form. The section of code starting with the label *in_mode1* checks to see if the pulses are coming in so slowly that both HSI lines can be checked. If this is the case then all of the HSIs are enabled and the program returns. This routine is the secondary method for going from mode 1 to mode 0, the primary method is by checking the time between edges during the HSI routine, which will be described later.

The HSO routine will enable mode 0 from mode 1 if two edges are not received every 260 microseconds. The primary method, (under the HSI routine), can only

enable mode 0 after an edge is received. This could cause a problem if the last 2 edges on Phase A before the encoder stops were too close to enable mode 0. If this happened, mode 0 would not be enabled until after the encoder started again, resulting in missed edges on Phase B. Using the HSO routine to switch from mode 1 to mode 0 eliminates this problem.

Figure 4-6 shows a state diagram of how the mode switching is done. As can be seen, there are two sources for most of the mode decisions. This helps avoid problems such as the one mentioned above.

When either Mode 1 or Mode 0 is enabled the HSI interrupt routine performs the counting of edges, while the HSO routine only ensures that the correct mode is running. The routines for modes 0 and 1 share the same initialization and completion sections, with the main body of code being different.

The initialization routine is similar to many HSI routines. The flags are checked to ensure that the HSI FIFO data is valid, and then the FIFO is read. Next, the main body of code (for either mode 0 or mode 1) is

run. At the end time and count values are saved and the holding register is checked for another event. Listing 4-8 contains the initialization and completion sections of the HSI routine.

Listing 4-9 is the main body of the Mode 1 routine. Before any calculations are done in Mode 1, the incoming pulse period is measured to see if it is too fast or too slow for mode 1. The time period between two edges is used so that the duty cycle of the waveform will not affect mode switching. If it is determined that Mode 2 should be set, Port 1.1 is set, all of the HSI lines are disabled, and the HSI fifo is cleared. If Mode 0 is to be set all of the HSI lines are enabled and the variable *LAST_STAT* is cleared. *LAST_STAT* = 0 is used as a flag to indicate the first HSI interrupt in Mode 0 after Mode 1. After the mode checking and setting are complete the incremental value in Timer 2 is used to update

POSITION. The program then returns to the completion section of the routine.

There is a lot more code used in Mode 0 than in Mode 1, most of which is due to the multiple jump statements that determine the current and previous state of the HSI pins. In order to save execution time several blocks of code are repeated as can be seen in Listing 4-10. The first determination is that of which edge had occurred. If a Phase A edge was detected the *LAST1_TIME* and *LAST2_TIME* variables are updated so a reference to the pulse frequency will be available. These are the same variables used under Mode 1. A test is also made to see if the edges are coming fast enough to warrant being in Mode 1, if they are, the switch is made. If the last edge detected was on Phase B, the information is used only to determine direction.

```

In_mode_1:                ; mode 1 HSI routine
    andb    tmp1, hsi_s0, #01010000B
    jne     no_cnt
    cmp_time:
        ld      last2_time, last1_time
        ld      last1_time, time
    cmpl:   sub     tmp1, time, last2_time
    cmp     tmp1, min_hsil
    jh      check_max_time

    set_mode_2:
        orb     Port1, #000000010B
        ldb     IOC0, #00000000B
    mt_hsi: ld      zero, hsi_time
        andb    iosl_bak, #01111111B
        orb     iosl_bak, iosl
        jbs     iosl_bak, 7, mt_hsi
        br      done_chk

    check_max_time:
        sub     tmp1, time, last2_time
        cmp     tmp1, max_hsil
        jnh     done_chk

    set_mode_0:
        andb    Port1, #11111100B
        ldb     IOC0, #01010101B
        ldb     last_stat, zero

    done_chk:
        sub     delta_p, timer_2, tmr2_old
        jbc     direct, 0, add_rev

    add_fwd:
        add     position, delta_p
        addc    position+2, zero
        load_last
    add_rev:
        sub     position, delta_p
        subc    position+2, zero
        load_last

    $eject

```

270061-54

Listing 4-9. Motor Control Mode 1 Routines

```

In_mode_0:
    jbs     hsi_s0,0,a_rise
    jbs     hsi_s0,2,a_fall
    jbs     hsi_s0,4,b_rise
    jbs     hsi_s0,6,b_fall
    br     no_cnt

a_rise:   ld     last2_time,last1_time
          ld     last1_time,time
          sub     time,last2_time
          cmp     time,min_hsi
          jh     tst_statf
          ;set model-
          orb     Port1,#000000001B      ; Set P1.0 (in mode 1)
          ldb     IOCO,#000000101B      ; Enable HSI 0 and 1
          tst_statf:
          jbs     last_stat,6,going_fwd
          jbs     last_stat,4,going_rev
          jbs     last_stat,2,change_dir
          cmpb    last_stat,zero
          je     first_time
          br     inp_err

a_fall:   ld     last2_time,last1_time
          ld     last1_time,time
          sub     time,last2_time
          cmp     time,min_hsi
          jh     tst_statf
          ;set model-
          orb     Port1,#000000001B      ; Set P1.0 (in mode 1)
          ldb     IOCO,#000000101B      ; Enable HSI 0 and 1
          tst_statf:
          jbs     last_stat,4,going_fwd
          jbs     last_stat,6,going_rev
          jbs     last_stat,0,change_dir
          cmpb    last_stat,zero
          je     first_time
          br     inp_err

b_rise:   jbs     last_stat,0,going_fwd
          jbs     last_stat,2,going_rev
          jbs     last_stat,6,change_dir
          cmpb    last_stat,zero
          je     first_time
          br     inp_err

b_fall:   jbs     last_stat,2,going_fwd
          jbs     last_stat,0,going_rev
          jbs     last_stat,4,change_dir
          cmpb    last_stat,zero
          je     first_time
          br     inp_err

first_time:
          stb     hsi_s0,last_stat
          br     done_chk                ; add delta position
inp_err:   br     no_int

change_dir:
          notb    direct
no_inc:   jbc     direct,0,going_rev

going_fwd:
          orb     PORT2,#01000000B      ; set P2.6
          ldb     direct,#01            ; direction = forward
          add     position,#01
          addc    position+2,zero
          br     st_stat

going_rev:
          andb    PORT2,#10111111B      ; clear P2.6
          ldb     direct,#00            ; direction = reverse
          sub     position,#01
          subc    position+2,zero

st_stat:   stb     hsi_s0,last_stat

```

270061-55

Listing 4-10. Motor Control Mode 0 Routines

After mode correctness is confirmed and the *LASTx_TIME* values are updated the *LAST_STAT* (Last Status) variable is used to determine the current direction of travel. The *POSITION* value is then updated in the direction specified by the last two edges and the status is stored. Note that the first time in Mode 0 after being in Mode 1, the Mode 1 *done_chk* routine is used to update *POSITION*, instead of the routines *going_fwd* and *going_rev* from the Mode 0 section of code. The completion section of code is then executed.

Providing the PWM value to drive the motor is done by a routine running under Software Timer 1. The first section of code, shown in Listing 4-11a, has to do with calculating the position and timer errors. Listing 4-11b shows the next section of code where the power to be supplied to the motor is calculated. First the direction is checked and if the direction is reverse the absolute value of the error is taken. If the error is greater than 64K counts, the PWM routine is loaded with the maximum value. The next check is made to see if the motor

is close enough to the desired location that the power to it should be reversed, (i.e., enter the Braking mode). If the motor is very close to the position or has slowed to the point that is likely to turn around, the *Hold_Position mode* is entered.

The determination of which modes are selected under what conditions was done empirically. All of the parameters used to determine the mode are kept in RAM so they can be easily changed on the fly instead of by re-assembling the program. The parameters in the listing have been selected to make the motor run, but have not been optimized for speed or stability. A diagram of the modes is shown in Figure 4-7.

In the *Hold_Position* mode power is eased onto the motor to lock it into position. Since the motor could be stopped in this mode, some integral control is needed, as proportional control alone does not work well when the error is small and the load is large. The *BOOST* variable provides this integral control by increasing the output a fixed amount every time period in which the

Listing 4-11. Motor Control Software Timer 1 Routine

```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!                                SOFTWARE TIMER ROUTINE 1                                !!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

CSEG AT 2600H                                ; set port1.7
swtl_expired:                                ; enable HSI, Tcvf, HSO
    pushf
    orb    port1,$10000000B
    ldb    int_mask,$00001101B
    ldb    HSO_COMMAND,$39H
    add    HSO_TIME,TIMER1,swtl_dly

    ld     time_err+2,des_time+2    ; Calculate time & position error
    ld     pos_err+2,des_pos+2
    sub    time_err,des_time,time    ; values are set
    subc   time_err+2,time+2
    sub    pos_err,des_pos,position
    subc   pos_err+2,position+2

    EI

    sub    time_delta,last_time_err,time_err
    ld     last_time_err,time_err

    sub    pos_delta,last_pos_err,pos_err
    ld     last_pos_err,pos_err

    ; Time_err = Desired time to finish - current time
    ; Pos_err = Desired position to finish - current position
    ; Pos_delta = Last position error - Current position error
    ; Time_delta = Last time error - Current time error
    ; note that errors should get smaller so deltas will be
    ; positive for forward motion (time is always forward)

```

270061-56

Listing 4-11a. Motor Control Software Position Counter

```

        cmp     pos_err+2,zero
        jge     go_forward
        go_backward:
            neg     pos_err
            ldb     pwm_dir,$000h
            cmp     pos_err+2,$00ffffH
            jne     ld_max
            br      chk_brk
        go_forward:
            ldb     pwm_dir,$001h
            cmp     pos_err+2,zero
            jle     chk_brk
            ldb     max_pwr,max_pwr
            br      chk_sanity
        chk_brk:
            cmp     pos_err,pos_pnt
            jnh     hold_position
            cmp     pos_err,brk_pnt
            jh     ld_max
        braking:
            cmp     pos_delta,zero
            jge     chk_delta
            neg     pos_delta
        chk_delta:
            cmp     pos_delta,vel_pnt
            jnh     hold_position
        brake:
            ldb     pwm_pwr,max_brk
            ldb     tmp,direct
            notb    tmp
            ldb     pwm_dir,tmp
            br      ld_pwr
        Hold_position:
            cmp     pos_err,$02
            jh     calc_out
            clr     tmp+2
            clr     boost
            BR      output
        calc_out:
            mulub   tmp,max_hold,$255
            mulu    tmp,pos_err
            cmp     pos_delta,zero
            jne     no_bst
            add     boost,$04
            add     tmp+2,boost
            br      ck_max
            no_bst:
            clr     boost
            ck_max:
            cmp     tmp+2,max_hold
            jnh     output
            maxed:
            ld      tmp+2,max_hold
            output:
            ldb     pwm_pwr,tmp+2
        chk_sanity:
            br      ld_pwr
        ld_pwr:
            ldb     rpwr,pwm_pwr
            notb    rpwr
            jbs     pwm_dir,0,p2fwd
        p2bkwd:
            DI      port2,$01111111B
            ldb     pwm_control,rpwr
            EI
            br      pwrset
        p2fwd:
            DI      port2,$10000000B
            ORB     port2,$10000000B
            ldb     pwm_control,rpwr
            EI

```

270061-57

Listing 4-11b: Motor Control Power Algorithm

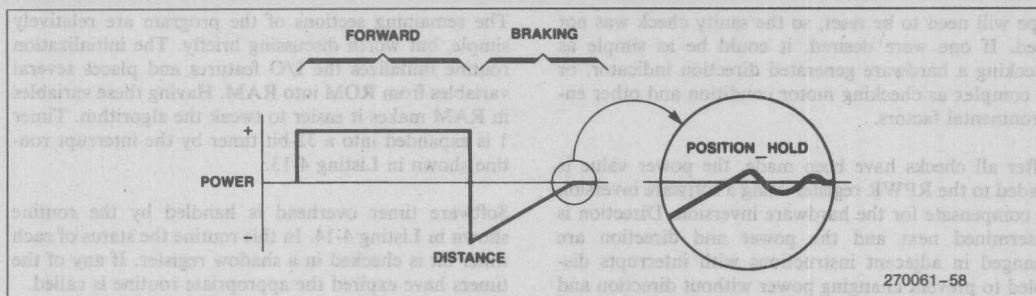


Figure 4-7. Motor Control Modes

error does not get smaller. Once the error does get smaller, usually because the motor starts moving, BOOST is cleared.

A sanity check can be performed at this point to double check that the 8096 has proper control of the motor. In the example the worst that can happen is the proto-

```

pwrset:      cmp     time_err+2,zero    ; do pos_table when err is negative
            jgt     end_p
            br      end_p

            cmp     nxt_pos,#(32+pos_table)
            jlt     get_vals            ; jump if lower
            ld      nxt_pos,#pos_table
            clr     time+2

get_vals:    ld      des_pos,[nxt_pos]+
            ld      des_pos+2,[nxt_pos]+
            ld      des_time+2,[nxt_pos]+
            ld      max_pwr,[nxt_pos]+
            ld      max_brk,max_pwr
            add     des_pos,offset
            addc    des_pos+2,zero
            sub     last_pos_err,des_pos,position

end_p:      andb    port1,#01111111B    ; clear P1.7

            popf
            ret

pos_table:

            dcl     00000000H           ; position 0
            dcw     0020H, 0080H        ; next time, power
            dcl     0000c000H           ; position 1
            dcw     0040H, 0040H        ; next time, power
            dcl     00000000H           ; position 2
            dcw     0060H, 00c0H        ; next time, power
            dcl     0FFFF8000H          ; position 3
            dcw     0080H, 0080H        ; next time, power

            dcl     00000800H           ; position 4
            dcw     0058H, 0080H        ; next time, power
            dcl     00003000H           ; position 5
            dcw     0070H, 00ffH        ; next time, power
            dcl     00000000H           ; position 6
            dcw     0090H, 00f0H        ; next time, power
            dcl     00000000H           ; position 7
            dcw     0091H, 00f0H        ; next time, power

```

Listing 4-12. Motor Control Next Position Lookup

type will need to be reset, so the sanity check was not used. If one were desired, it could be as simple as checking a hardware generated direction indicator, or as complex as checking motor condition and other environmental factors.

After all checks have been made, the power value is loaded to the RPWR register using a software inversion to compensate for the hardware inversion. Direction is determined next and the power and direction are changed in adjacent instructions with interrupts disabled to prevent changing power without direction and vice versa.

To exercise the program logic the desired position is changed based on the time value using the code and lookup table shown in Listing 4-12.

The remaining sections of the program are relatively simple, but worth discussing briefly. The initialization routine initializes the I/O features and places several variables from ROM into RAM. Having these variables in RAM makes it easier to tweak the algorithm. Timer 1 is expanded into a 32-bit timer by the interrupt routine shown in Listing 4-13.

Software timer overhead is handled by the routine shown in Listing 4-14. In this routine the status of each timer bit is checked in a shadow register. If any of the timers have expired the appropriate routine is called.

```

////////////////////////////////////
TIMER INTERRUPT SERVICE
////////////////////////////////////

CSEG AT 2200H

timer_ovf_int:
    pushf
    orb     iosl_bak,IOS1
    jbc     iosl_bak,5,tmr_int_done
    inc     time+2
    andb    iosl_bak,$11011111B    ; clear bit 5
    tmr_int_done:
    popf
    ret                                ; End of timer interrupt routine
    
```

270061-60

Listing 4-13. Motor Control Timer Interrupt Routine

```

////////////////////////////////////
SOFTWARE TIMER INTERRUPT SERVICE ROUTINE
////////////////////////////////////

CSEG AT 2220H

soft_tmr_int:
    pushf
    orb     iosl_bak,IOS1
    chk_swt0:
    jbc     iosl_bak,0,chk_swt1
    andb    iosl_bak,$11111110B    ; Clear bit 0 - end swt0
    call    swt0_expired
    chk_swt1:
    jbc     iosl_bak,1,chk_swt2
    andb    iosl_bak,$11111101B    ; Clear bit 1
    call    swt1_expired
    chk_swt2:
    jbc     iosl_bak,2,chk_swt3
    andb    iosl_bak,$11111011B    ; Clear bit 2
    call    swt2_expired
    chk_swt3:
    jbc     iosl_bak,4,swt_int_done
    andb    iosl_bak,$11110111B    ; Clear bit 3
    call    swt3_expired

    swt_int_done:
    popf
    ret                                ; END OF SOFTWARE TIMER INTERRUPT ROUTINE

$seject
    
```

270061-B2

Listing 4-14. Motor Control Software Timer Interrupt Handler

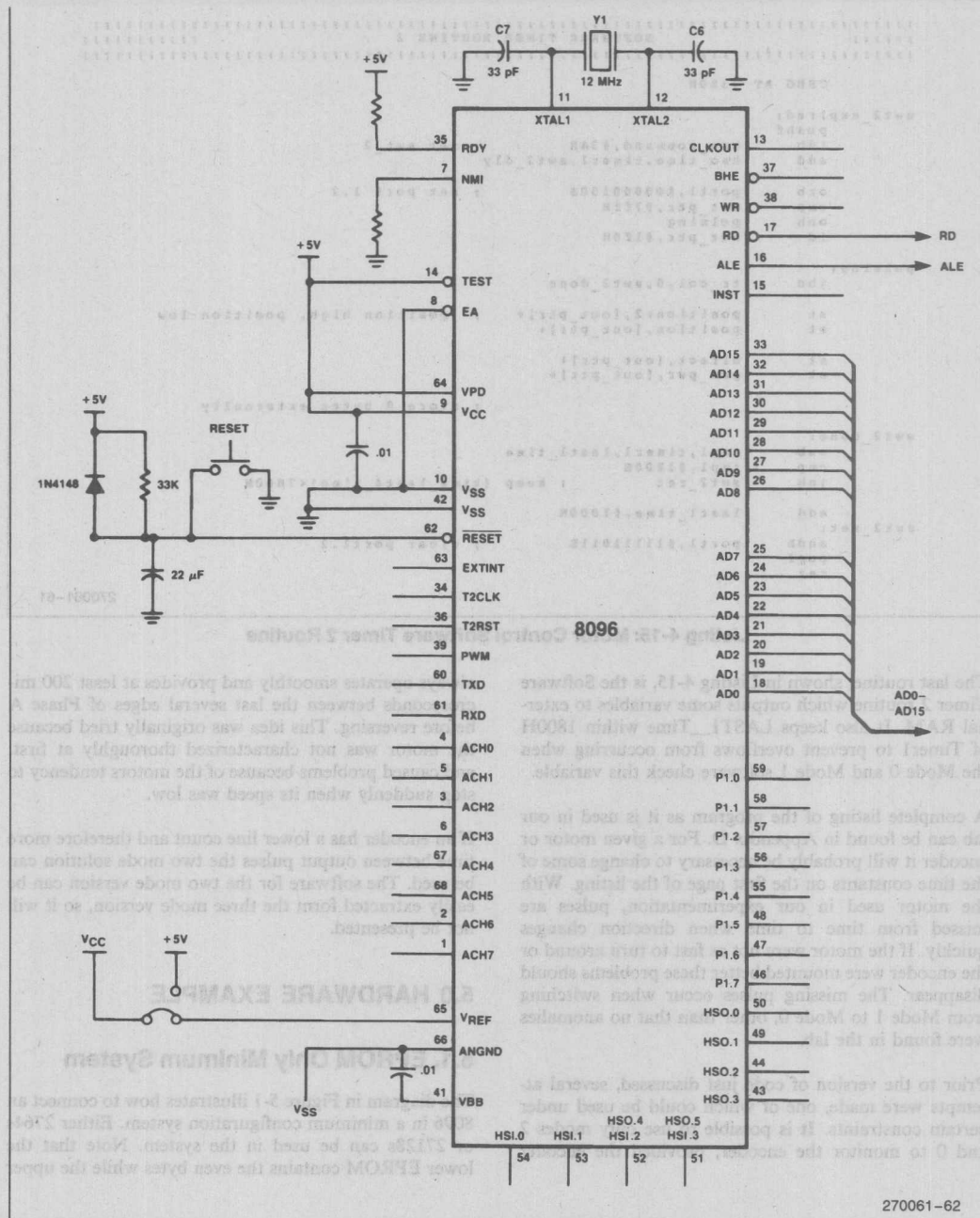


Figure 5-1 (1 of 2).

one contains the odd bytes, and the addressing is not fully decoded. This means that the addressing on a 2764 will be such that the lower 4K of each EPROM is mapped at 0000H and 4000H while the upper

4K is mapped at 2000H. If the program being loaded is 16 Kbytes long the first half is loaded into the second half of the 2764s and vice versa. A similar situation exists when using 27128s.

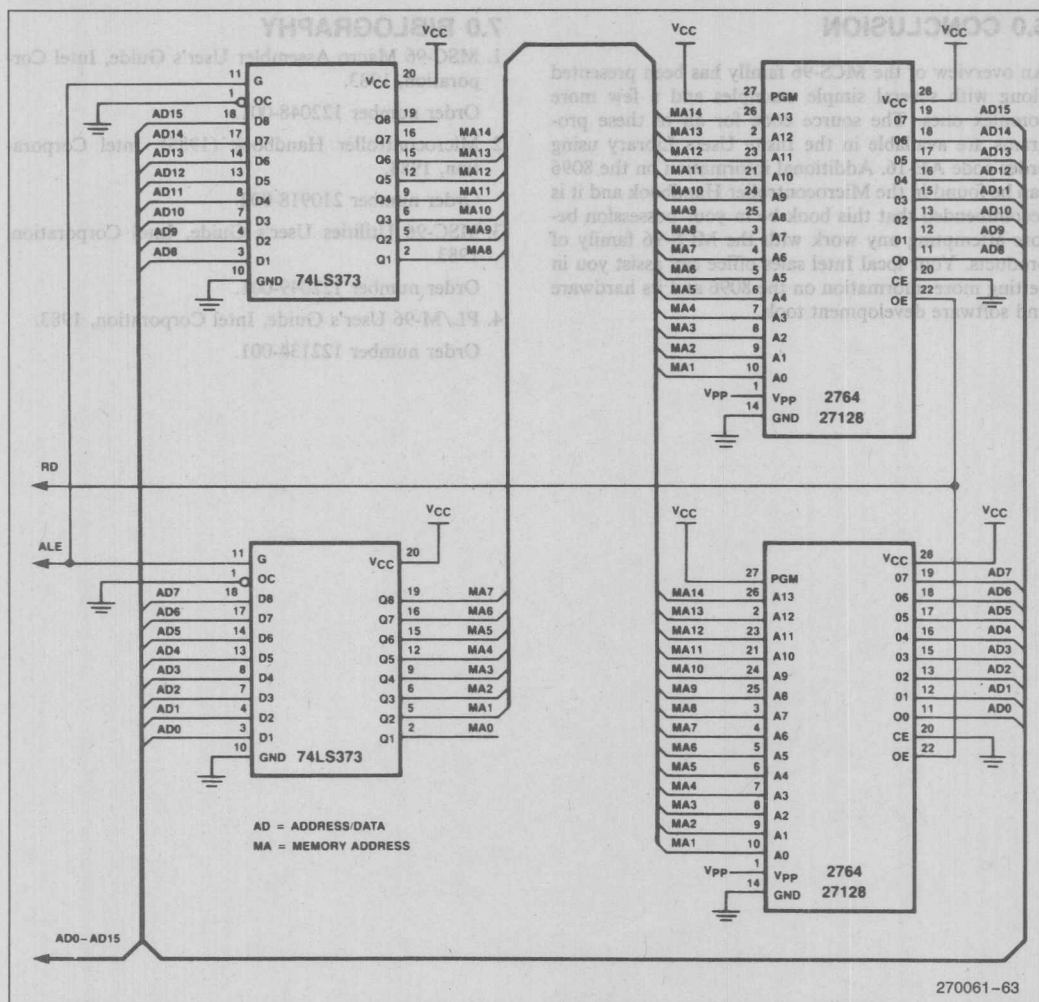


Figure 5-1 (2 of 2).

This circuit will allow most of the software presented in this ap-note to be run. In a system designed for prototyping in the lab it may be desirable to buffer the I/O ports to reduce the risk of burning out the chip during experimentation. One may also want to enhance the system by providing RC filters on the A to D inputs, a precision VREF power supply, and additional RAM.

5.2. Port Reconstruction

If it is desired to fully emulate a 8396 then I/O ports 3 and 4 must be reconstructed. It is easiest to do this if

the usage of the lines can be restricted to inputs or outputs on a port by port rather than line by line basis. The ports are reconstructed by using standard memory-mapped I/O techniques, (i.e., address decoders and latches), at the appropriate addresses. If no external RAM is being used in the system then the address decoding can be partial, resulting in less complex logic.

The reconstructed I/O ports will work with the same code as the on chip ports. The only difference will be the propagation delay in the external circuitry.

6.0 CONCLUSION

An overview of the MCS-96 family has been presented along with several simple examples and a few more complex ones. The source code for all of these programs are available in the Insite Users Library using order code AE-16. Additional information on the 8096 can be found in the Microcontroller Handbook and it is recommended that this book be in your possession before attempting any work with the MCS-96 family of products. Your local Intel sales office can assist you in getting more information on the 8096 and its hardware and software development tools.

7.0 BIBLIOGRAPHY

1. MSC-96 Macro Assembler User's Guide, Intel Corporation, 1983.
Order number 122048-001.
2. Microcontroller Handbook (1985), Intel Corporation, 1984.
Order number 210918-002.
3. MSC-96 Utilities User's Guide, Intel Corporation, 1983.
Order number 122049-001.
4. PL/M-96 User's Guide, Intel Corporation, 1983.
Order number 122134-001.

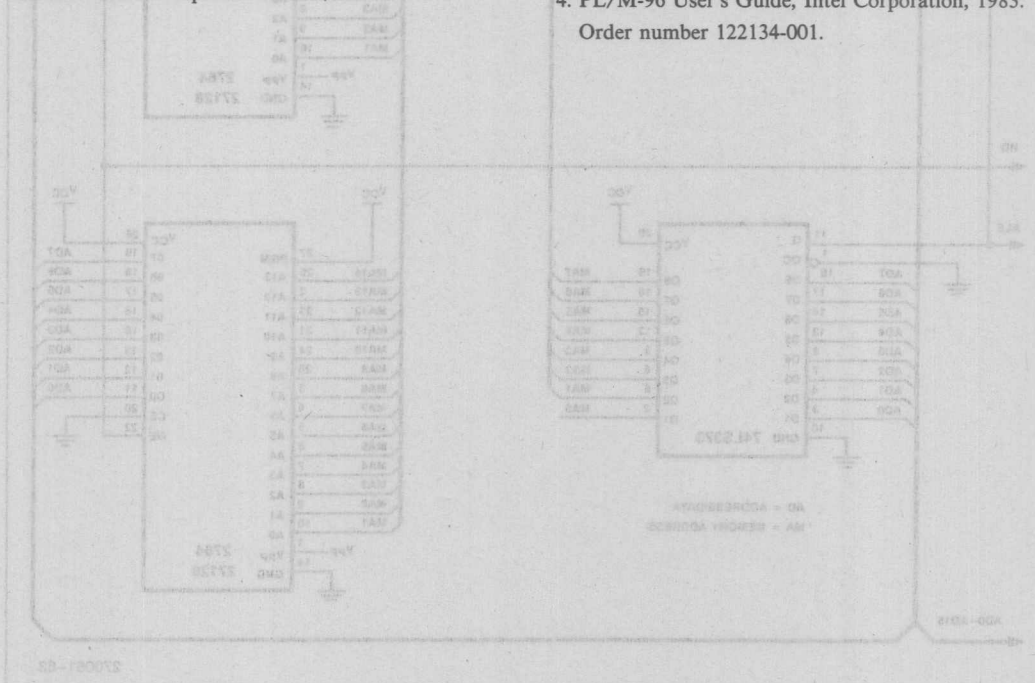


Figure 2-1 (S of 2)

The 8096 will allow most of the software presented in this report to be run in a system designed for prototyping in the lab. It may be desirable to buffer the I/O ports to reduce the risk of burning out the chip during experimentation. One may also want to enhance the system by providing ROM filters on the A to D inputs, a precision VREF power supply, and additional RAM.

The reconstructed I/O ports will work with the same code as the on-chip ports. The only difference will be the propagation delay in the external circuitry.

8.2 Port Reconstruction

If it is desired to fully emulate a 8096 then I/O ports 3 and 4 must be reconstructed. It is easiest to do this if

APPENDIX A BASIC SOFTWARE EXAMPLES

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F3:INTER1.A96

OBJECT FILE: F3:INTER1.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			1	%TITLE('INTER1.A96: Interpolation routine 1')
			2	;;;;;;;; 8096 Assembly code for table lookup and interpolation
			3	
			4	%INCLUDE('F0:DEMO96.INC') ; Include demo definitions
		=1	5	%nolist ; Turn listing off for include file
		=1	53	; End of include file
			54	
	0022		55	RSEG at 22H
	3103		56	END
	0022		57	IN_VAL: dsb 1 ; Actual Input Value
	0024 0070		58	TABLE_LOW: dsb 1
	0026 002000+0003+0055		59	TABLE_HIGH: dsb 1
	0028 0018001000100000		60	IN_DIF: dsb 1 ; Upper Input - Lower Input
	0028 0000000000150010		61	IN_DIFB equ IN_DIF byte
	002A 000000000002+00+0		62	TAB_DIF: dsb 1 ; Upper Output - Lower Output
	002C		63	OUT: dsb 1
	002E		64	RESULT: dsb 1
	0030		65	OUT_DIF: dsb 1 ; Delta Out
			66	
	2080		67	
			68	CSEG at 2080H
			69	
	2080 A1000118		70	LD SP, #100H
			71	
	2084 B0221C		72	look: LDB AL, IN_VAL ; Load temp with Actual Value
	2087 18031C		73	SHRB AL, #3 ; Divide the byte by 8
	208A 71FE1C		74	ANDB AL, #11111110B ; Insure AL is a word address
			75	; This effectively divides AL by 2
			76	; so AL = IN_VAL/16
			77	
	208D AC1C1C		78	LDBZ AX, AL ; Load byte AL to word AX
	2090 A31D002124		79	LD TABLE_LOW, TABLE[AX] ; TABLE_LOW is loaded with the value
			80	; in the table at table location AX
			81	

270061-64

A.1. Table Lookup 1 (Continued)

```

2095 A31D022126      82      LD      TABLE_HIGH, (TABLE+2)(AX) ; TABLE_HIGH is loaded with the
83                                     ; value in the table at table
84                                     ; location AX+2
85                                     ; (The next value in the table)
86
209A 4824262A      87      SUB      TAB_DIF, TABLE_HIGH, TABLE_LOW
88                                     ; TAB_DIF=TABLE_HIGH-TABLE_LOW
89
209E 510F222B      90      ANDB     IN_DIFB, IN_VAL, #0FH ; IN_DIFB=least significant 4 bits
91                                     ; of IN_VAL
20A2 AC2B2B      92      LDBZE    IN_DIF, IN_DIFB ; Load byte IN_DIFB to word IN_DIF
93
20A5 FE4C2A2B30      94      MUL      OUT_DIF, IN_DIF, TAB_DIF
95                                     ; Output_difference =
96                                     ; Input_difference*Table_difference
20AA 0E0430      97      SHRAL    OUT_DIF, #4 ; Divide by 16 (2**4)
98
20AD 4424302C      99      ADD      OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
100                                     ; generated with truncated IN_VAL
101                                     ; as input
20B1 0A042C      102     SHRA     OUT, #4 ; Round to 12-bit answer
20B4 A4002C      103     ADDC     OUT, zero ; Round up if Carry = 1
104
20B7 C02E2C      105     no_inc: ST     OUT, RESULT ; Store OUT to RESULT
106
20BA 27C8      107     C0C0 7F BR     look ; Branch to "look."
108
2100      109
110     cseg      AT 2100H
111
2100 000000200034004C      112     table: DCW     0000H, 2000H, 3400H, 4C00H ; A random function
2108 005D006A0072007B      113     DCW     5D00H, 6A00H, 7200H, 7B00H
2110 007B007D0076006D      114     DCW     7B00H, 7D00H, 7600H, 6D00H
2118 005D004B00340022      115     DCW     5D00H, 4B00H, 3400H, 2200H
2120 0010      116     DCW     1000H
2122      117
118     END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

270061-65

END FOR 087EC1

COMMENTS SPECIFIED IN INNOVATION COMMAND WORDS

OR'ECT LIFE: 10 INCHES 087

SOURCE LIFE: 10 INCHES 087

SERIES-111 ACB-30 INCHES 087

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F3.INTER2.A96

OBJECT FILE: F3.INTER2.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```

ERR LOC  OBJECT          LINE      SOURCE STATEMENT
1          $TITLE('INTER2.A96: Interpolation routine 2')
2
3          ; ; ; ; ; 8096 Assembly code for table lookup and interpolation
4          ; ; ; ; ; Using tabled values in place of division
5
6          $INCLUDE('F0.DEMO96.INC'), Include demo definitions
=1         7          $nolist ; Turn listing off for include file
=1         55         ; End of include file
56
VDEF 0024 COMPLETED 100 BUNCH(2) 57 RSEG at 24H
58
0024         59         IN_VAL:      dsb      1          ; Actual Input Value
0026         60         TABLE_LOW: dsb      1          ; Table value for function
002B 004E40E0E0E0E0E0 61         TABLE_INC: dsb 1600H 00E0H 00E0H Incremental change in function
002A 000040E010E00000 62         IN_DIF:   dsb 1600H 10E0H 00E0H Upper Input - Lower Input
5 002A 0000000000000000 63         IN_DIFB:  dsb 1600H IN_DIF: byte
002C 0000000000000000 64         OUT:      dsb 1600H 10E0H 00E0H
002E         65         RESULT:     dsb      1
0030 0010         66         OUT_DIF: dsb 1600H ds1      1          ; Delta Out
5110 0000000000000000 67         DCB      2000H 4000H 3400H 5000H
5116 0010001000100000 68         DCB      1800H 1000H 1900H 4000H
2080 0020000000100010 69         CSEG at 2080H 2000H 4000H 1500H 1800H
5100 0000000000000000 70         DCB      0000H 3000H 3400H 4000H
2080 A100011B         71         LD      SP, #100H ; Initialize SP to top of reg file
72
2084 B0241C         73         look:  LDB 100H AL, IN_VAL ; Load temp with Actual Value
2087 18031C         74         SHRB AL, #3 ; Divide the byte by 8
208A 71FE1C         75         ANDB AL, #1111110B ; Insure AL is a word address
5023 0000         76         SHL     AL, 1 ; This effectively divides AL by 2
5040 0000         77         LD      AL, 0 ; so AL = IN_VAL/16
208D AC1C1C         78         LDBZE AX, AL ; Load byte AL to word AX
509D 000000         79         VDDC 001 1000 ;
2090 A31D002126     80         LD      TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value
81 ; in the value table at location AX
82 ;
2095 A31D222128     83         LD      TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value
84 ; in the increment table at
85 ; location AX+2
86 ;
5091 0000000000000000 87         MOV     DI, DI ; IN DI, IN DI
88         88         FDB 10 10 10 10 ; 1000 0000 10 10 10 10 10 10
89         89         00 10 10 10 ; 00 10 10 10
5094 01003450         90         WDB 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

```

270061-66

```

209A 510F242A      87      ANDB     IN_DIFB, IN_VAL, #0FH      ; IN_DIFB=least significant 4 bits
209E AC2A2A        88                      ; of IN_VAL
209E AC2A2A        89      LDBZ     IN_DIF, IN_DIFB      ; Load byte IN_DIFB to word IN_DIF
20A1 FE4C2B2A30    90                      ;
20A1 FE4C2B2A30    91      MUL      OUT_DIF, IN_DIF, TABLE_INC ; Output_difference =
20A1 FE4C2B2A30    92                      ; Input_difference*Incremental_change
20A1 FE4C2B2A30    93                      ;
20A1 FE4C2B2A30    94                      ;
20A6 4426302C      95      ADD      OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
20A6 4426302C      96                      ; generated with truncated IN_VAL
20A6 4426302C      97                      ; as input
20AA 08042C        98      SHR      OUT, #4      ; Round to 12-bit answer
20AD A4002C        99      ADDC     OUT, zero      ; Round up if Carry = 1
20B0 C02E2C        100     no_inc: ST     OUT, RESULT ; Store OUT to RESULT
20B3 27CF          101     BR      look      ; Branch to "look:"
20B3 27CF          102     look:  ;
20B3 27CF          103     ;
20B3 27CF          104     ;
2100 B0581C        105     cseg   AT 2100H      ;
2100 B0581C        106     ;
2100 B0581C        107     val_table: ;
2100 B0581C        108     DCW     0000H, 2000H, 3400H, 4C00H ; A random function
2108 005D006A00720078 109     DCW     5D00H, 6A00H, 7200H, 7800H
2110 007B007D0076006D 110     DCW     7B00H, 7D00H, 7600H, 6D00H
2118 005D004B00340022 111     DCW     5D00H, 4B00H, 3400H, 2200H
2120 0010          112     DCW     1000H
2122          113     inc_table: ;
2122 0002400180011001 114     DCW     0200H, 0140H, 0180H, 0110H ; Table of incremental
212A D000800060003000 115     DCW     00D0H, 00B0H, 0060H, 0030H ; differences
2132 200090FF70FF00FF 116     DCW     00020H, 0FF90H, 0FF70H, 0FF00H
213A E0FE90FEE0FEE0FE 117     DCW     0FE00H, 0FE90H, 0FEE0H, 0FEE0H
2142          118     ;
2142          119     END

```

ASSEMBLY COMPLETED. NO ERROR(S) FOUND.

270061-67

END FOR OBJECT

FILE

SOURCE STATEMENT

CONSIDER SPECIALIZED IN INNOVATION COMMAND MORE

OBJECT LIFE IS INHERITORS

SOURCE LIFE IS INHERITORS

SERIES III W3-AT MICRO V204515A AT 0


```

20 1      IF CARRY=0 THEN RESULT=OUT; /* Using the hardware flags must be done */
22 1      ELSE RESULT=OUT+1;          /* with care to ensure the flag is tested */
23 1      GOTO LOOP;                  /* in the desired instruction sequence */

```

```
/* END OF PLM-96 CODE */
```

```

24 1      END;

```

270061-69

PL/M-96 COMPILER PLMEX1: PLM-96 Example Code for Table Lookup
ASSEMBLY LISTING OF OBJECT CODE

```

10 1      0022          PLMEX:
12 1      0022 A1000018      LD SP,#STACK
14 1      0026          LOOP:
16 1      0026 A00010      LD TEMP,IN_VAL
18 1      0029 0B0410      SHR TEMP,#4H
20 1      002C 4410101C      ADD TMP0,TEMP,TEMP
22 1      0030 A31D000002      LD TABLE_LOW,TABLE[TMP0]
24 1      0035 A31D020004      LD TABLE_HIGH,TABLE+2H[TMP0]
26 1      003A 4B020406      SUB TABLE_DIF,TABLE_HIGH,TABLE_LOW
28 1      003E 0C806      PUSH TABLE_DIF
30 1      0040 410F00001C      AND TMP0,IN_VAL,#0FH
32 1      0045 0CB1C      PUSH TMP0
34 1      0047 0EF0000      CALL DMPY
36 1      004A 0E041C      SHRAL TMP0,#4H
38 1      004D 0A01E0E      LD OUT_DIF+2H,TMP2
40 1      0050 0A01C0C      LD OUT_DIF,TMP0
42 1      0053 A00220      LD TMP4,TABLE_LOW
44 1      0056 0620      EXT TMP4
46 1      0058 641C20      ADD TMP4,TMP0
48 1      005B A41E22      ADDC TMP6,TMP2
50 1      005E 0E0420      SHRAL TMP4,#4H
52 1      0061 A02008      LD OUT,TMP4
54 1      0064 B1FF1C      LDB TMP0,#OFFH
56 1      0067 DB02      BC @0003
58 1      0069 111C      CLRB TMP0
60 1      006B          @0003:

```

270061-70

```

006B 9B1C00      CMPB  RO, TMP0
006E D705        BNE   @0001
                ; STATEMENT 21
0070 A0200A      LD     RESULT, TMP4
0073 2005        BR     @0002
                ; STATEMENT 22
0075             @0001:
0075 A0080A      R      LD     RESULT, OUT
007B 070A        R      INC    RESULT
                ; STATEMENT 23
007A             @0002:
007A 27AA        BR     LOOP
                ; STATEMENT 24
                END

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 005AH  90D
CONSTANT AREA SIZE  = 0022H  34D
DATA AREA SIZE      = 0000H   0D
STATIC REGS AREA SIZE = 0012H  18D

```

```

PL/M-96 COMPILER  PLMEX1 = PLM-96 Example Code for Table Lookup
                    ASSEMBLY LISTING OF OBJECT CODE

```

```

OVERLAYABLE REGS AREA SIZE = 0000H  0D
MAXIMUM STACK SIZE        = 0006H  6D
48 LINES READ

```

```

PL/M-96 COMPILATION COMPLETE  0 WARNINGS, 0 ERRORS

```

270061-71

MCS-96 MACRO ASSEMBLER MULT.APT: 16*16 multiply procedure for PLM-96

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:MULT.A96

OBJECT FILE: :F3:MULT.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	\$TITLE('MULT.APT: 16*16 multiply procedure for PLM-96')
		2	
		3	
0018		4	SP EGU 1BH:word
		5	
0000		6	rseg
		7	EXTRN PLMREG :long
		8	
0000		9	cseg
		10	
		11	PUBLIC DMPY ; Multiply two integers and return a
		12	; longint result in AX, DX registers
		13	
0000 CC04	E	14	DMPY: POP PLMREG+4 ; Load return address
0002 CC00	E	15	POP PLMREG ; Load one operand
0004 FE6E1900	E	16	MUL PLMREG,[SP]+ ; Load second operand and increment SP
		17	
0008 E304	E	18	BR [PLMREG+4] ; Return to PLM code.
000A		19	END.

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

CONSUMED VARY SIZE = 0035H 3+0

CODE VARY SIZE = 0024H 300

270061-72

SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation

INPUT FILES: :F3:PLMEX1.OBJ, :F3:MULT.OBJ, PLM96.LIB
OUTPUT FILE: :F3:PLMOUT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:
ROM(2080H-3FFFH)

INPUT MODULES INCLUDED:
:F3:PLMEX1.OBJ(PLMEX) 12/25/84
:F3:MULT.OBJ(MULT) 12/25/84
PLM96.LIB(PLMREG) 11/02/83

SEGMENT MAP FOR :F3:PLMOUT.OBJ(PLMEX):

TYPE	BASE	LENGTH	ALIGNMENT	MODULE NAME
**RESERVED*	0000H	001AH		
*** GAP ***	001AH	0002H		
REG	001CH	0008H	ABSOLUTE	PLMREG
REG	0024H	0012H	WORD	PLMEX
STACK	0036H	0006H	WORD	
*** GAP ***	003CH	2044H		
CODE	2080H	0003H	ABSOLUTE	PLMEX
*** GAP ***	2083H	0001H		
CODE	2084H	007CH	WORD	PLMEX
CODE	2100H	000AH	BYTE	MULT
*** GAP ***	210AH	DEF6H		

270061-73

WTT WTT 7FC4H MEMOIA BIEE
WTT WTT 003CH MEMOIA
WEO FONG 003CH 67H8E3
CODE DATA 2100H DATA
WEO MEND 003CH 10H
WEO FONGHIL 003CH 001 DIL
WEO INLESEN 003CH 87001
WEO INLESEN 003CH 001
WEO INLESEN 003CH 1001E DIL
WEO INLESEN 003CH 1001E HIGH
WEO INLESEN 003CH 1001E LOW
WEO MEND 003CH 10 HWT
6001E3

WTT WTT 7FC4H MEMOIA BIEE
WTT WTT 003CH MEMOIA
WEO FONG 003CH 67H8E3
CODE DATA 2100H DATA
WEO MEND 003CH 10H
WEO FONGHIL 003CH 001 DIL
WEO INLESEN 003CH 87001
WEO INLESEN 003CH 001
WEO INLESEN 003CH 1001E DIL
WEO INLESEN 003CH 1001E HIGH
WEO INLESEN 003CH 1001E LOW
WEO MEND 003CH 10 HWT
6001E3

WTT WTT 7FC4H MEMOIA BIEE
WTT WTT 003CH MEMOIA
WEO FONG 003CH 67H8E3
CODE DATA 2100H DATA
WEO MEND 003CH 10H
WEO FONGHIL 003CH 001 DIL
WEO INLESEN 003CH 87001
WEO INLESEN 003CH 001
WEO INLESEN 003CH 1001E DIL
WEO INLESEN 003CH 1001E HIGH
WEO INLESEN 003CH 1001E LOW
WEO MEND 003CH 10 HWT
6001E3

SYMBOL TABLE FOR :F3:PLMOUT.OBJ(PLMEX).

ATTRIBUTES	VALUE	NAME
REG	WORD	0024H
REG	INTEGER	0026H
REG	INTEGER	0028H
REG	INTEGER	002AH
REG	INTEGER	002CH
REG	INTEGER	002EH
REG	LONGINT	0030H
REG	WORD	0034H
CODE	ENTRY	2100H
REG	LONG	001CH
NULL	NULL	003CH
NULL	NULL	1FC4H

PUBLICS:

IN_VAL
TABLE_LOW
TABLE_HIGH
TABLE_DIF
OUT
RESULT
OUT_DIF
TEMP
DMPY
PLMREG
MEMORY
?MEMORY_SIZE

MODULE: PLMEX

MODULE: MULT

MODULE: PLMREG

```

*** CVA ***
CODE 3100H 0000
CODE 3080H 0000
*** CVA ***
CODE 3080H 0000
*** CVA ***
CODE 3080H 0000
RL96 COMPLETED, 0 WARNING(S), 0 ERROR(S)
REG 0034H 0015H
REG 001CH 0000H
*** CVA ***
*** DELETED ***
0015H 0000H
0015H 0000H

```

270061-74

TABLE 3100H 0000H 0000H 0000H 0000H

SEGMENT 1000 1000 1000 1000 1000

1000 1000 1000 1000 1000
1000 1000 1000 1000 1000
1000 1000 1000 1000 1000
1000 1000 1000 1000 1000

1000 1000 1000 1000 1000
1000 1000 1000 1000 1000
1000 1000 1000 1000 1000
1000 1000 1000 1000 1000

1000 1000 1000 1000 1000
1000 1000 1000 1000 1000
1000 1000 1000 1000 1000
1000 1000 1000 1000 1000

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE F3:PULSE.A96

OBJECT FILE F3:PULSE.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND NOSB

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	\$TITLE('PULSE.A96: Measuring pulses using the HSI unit')
		2	
		3	\$INCLUDE(DEMO96.INC)
=1		4	\$nolist ; Turn listing off for include file
=1		52	; End of include file
		53	
	002B	54	rseg at 28H
		55	
	002B	56	HIGH_TIME: dsw 1
	002A	57	LOW_TIME: dsw 1
	002C	58	PERIOD: dsw 1
	002E	59	HI_EDGE: dsw 1
	0030	60	LO_EDGE: dsw 1
		61	
		62	
		63	
	2080	64	cseg at 2080H
		65	
		66	
	2080 A100011B	67	LD SP, #100H
	2084 B10115	68	LDB IOCO, #00000001B ; Enable HSI 0
	2087 B10F03	69	LDB HSI_MODE, #00001111B ; HSI 0 look for either edge
		70	
	208A 442A282C	71	wait: ADD PERIOD, HIGH_TIME, LOW_TIME
	208E 3E1603	72	JBS IOS1, 6, contin ; If FIFO is full
	2091 3716F6	73	JBC IOS1, 7, wait ; Wait while no pulse is entered
		74	
	2094 B0061C	75	contin: LDB AL, HSI_STATUS ; Load status; Note that reading
		76	; HSI_TIME clears HSI_STATUS
		77	
	2097 A00420	78	LD BX, HSI_TIME ; Load the HSI_TIME
		79	
	209A 391C09	80	JBS AL, 1, hsi_hi ; Jump if HSI 0 is high
		81	
	209D C03020	82	hsi_lo: ST BX, LO_EDGE
	20A0 4B2E302B	83	SUB HIGH_TIME, LO_EDGE, HI_EDGE
	20A4 27E4	84	BR wait
		85	
		86	
	20A6 C02E20	87	hsi_hi: ST BX, HI_EDGE
		88	
		89	
		90	
		91	
		92	
		93	
		94	
		95	
		96	
		97	
		98	
		99	
		100	

270061-75

270061-76

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE F3 ENHSI A96

OBJECT FILE F3 ENHSI OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND NOSB

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	\$TITLE ('ENHSI A96 ENHANCED HSI PULSE ROUTINE')
		2	
		3	\$INCLUDE(Demo96.INC)
		4	\$nolist ; Turn listing off for include file
		52	; End of include file
		53	
0028		54	RSEG AT 28H
		55	
0028		56	TIME: DSW 1
002A		57	LAST_RISE: DSW 1
002C		58	LAST_FALL: DSW 1
002E		59	HSI_SO: DSB 1
002F		60	IOS1_BAK: DSB 1
0030		61	PERIOD: DSW 1
0032		62	LOW_TIME: DSW 1
0034		63	HIGH_TIME: DSW 1
0036		64	COUNT: DSW 1
		65	
2080		66	cseg at 2080H
		67	
2080 A100011B		68	init: LD SP,#100H
		69	
2084 B12516		70	LDB IOC1,#00100101B ; Disable HSD.4,HSD.5, HSI_INT=first,
		71	; Enable PWM,TXD,TIMER1_OVRFLOW_INT
		72	
2087 B19903		73	LDB HSI_MODE,#10011001B ; set hsi.1-; hsi.0 +
208A B10715		74	LDB IOC0,#00000111B ; Enable hsi.0,1
		75	; T2_CLOCK=T2CLK, T2RST=T2RST
		76	; Clear timer2
		77	
		78	
208D 717F2F		79	wait: ANDB IOS1_BAK,#01111111B ; Clear IOS1_BAK.7
2090 90162F		80	ORB IOS1_BAK,IOS1 ; Store into temp to avoid clearing
		81	; other flags which may be needed
2093 372FF7		82	JBC IOS1_BAK,7,wait ; If hsi is not triggered then
		83	; jump to wait
		84	
2096 5155062E		85	ANDB HSI_SO,HSI_STATUS,#01010101B
209A A0042B		86	LD TIME,HSI_TIME
		87	

270061-77

```

209D 382E05      88      JBS      HSI_SO,0,a_rise
20A0 3A2E0F      89      JBS      HSI_SO,2,a_fall
20A3 201A        90      BR       no_cnt

20A5 482C2832    92      a_rise: SUB    LOW_TIME, TIME, LAST_FALL
20A9 482A2830    93      SUB     PERIOD, TIME, LAST_RISE
20AD A0282A      94      LD       LAST_RISE, TIME
20B0 200B        95      BR       increment
                96
20B2 482A2834    97      a_fall: SUB    HIGH_TIME, TIME, LAST_RISE
20B6 482C2830    98      SUB     PERIOD, TIME, LAST_FALL
20BA A0282C      99      LD       LAST_FALL, TIME
                100
20BD            101      increment:
20BD 0736        102      INC     COUNT
20BF 27CC        103      no_cnt: BR    wait
                104
20C1            105      END
ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

20B4 51521F      37      JBR       10C1 8001001018
20B0 51000118    38      JBR       85:STOCH
                39
20B0            40      C=88    91      5080H
                41
203R            42      CORN1    DCM 1
203R            43      HIGH TIME DCM 1
203S            44      LOW TIME DCM 1
2030            45      LEN100 DCM 1
205L            46      1001 SWM DCM 1
205E            47      H21 SO DCM 1
205C            48      1001 LMT DCM 1
205V            49      1001 WISE DCM 1
205B            50      LINE DCM 1
                51
205B            52      5200 V1 50H
                53
                54      20
                55      1
                56      3
                57      5
                58      1
                59      1
                60      1
                61      1
                62      1
                63      1
                64      1
                65      1
                66      1
                67      1
                68      1
                69      1
                70      1
                71      1
                72      1
                73      1
                74      1
                75      1
                76      1
                77      1
                78      1
                79      1
                80      1
                81      1
                82      1
                83      1
                84      1
                85      1
                86      1
                87      1
                88      1
                89      1
                90      1
                91      1
                92      1
                93      1
                94      1
                95      1
                96      1
                97      1
                98      1
                99      1
                100      1
                101      1
                102      1
                103      1
                104      1
                105      1
                106      1
                107      1
                108      1
                109      1
                110      1
                111      1
                112      1
                113      1
                114      1
                115      1
                116      1
                117      1
                118      1
                119      1
                120      1
                121      1
                122      1
                123      1
                124      1
                125      1
                126      1
                127      1
                128      1
                129      1
                130      1
                131      1
                132      1
                133      1
                134      1
                135      1
                136      1
                137      1
                138      1
                139      1
                140      1
                141      1
                142      1
                143      1
                144      1
                145      1
                146      1
                147      1
                148      1
                149      1
                150      1
                151      1
                152      1
                153      1
                154      1
                155      1
                156      1
                157      1
                158      1
                159      1
                160      1
                161      1
                162      1
                163      1
                164      1
                165      1
                166      1
                167      1
                168      1
                169      1
                170      1
                171      1
                172      1
                173      1
                174      1
                175      1
                176      1
                177      1
                178      1
                179      1
                180      1
                181      1
                182      1
                183      1
                184      1
                185      1
                186      1
                187      1
                188      1
                189      1
                190      1
                191      1
                192      1
                193      1
                194      1
                195      1
                196      1
                197      1
                198      1
                199      1
                200      1
                201      1
                202      1
                203      1
                204      1
                205      1
                206      1
                207      1
                208      1
                209      1
                210      1
                211      1
                212      1
                213      1
                214      1
                215      1
                216      1
                217      1
                218      1
                219      1
                220      1
                221      1
                222      1
                223      1
                224      1
                225      1
                226      1
                227      1
                228      1
                229      1
                230      1
                231      1
                232      1
                233      1
                234      1
                235      1
                236      1
                237      1
                238      1
                239      1
                240      1
                241      1
                242      1
                243      1
                244      1
                245      1
                246      1
                247      1
                248      1
                249      1
                250      1
                251      1
                252      1
                253      1
                254      1
                255      1
                256      1
                257      1
                258      1
                259      1
                260      1
                261      1
                262      1
                263      1
                264      1
                265      1
                266      1
                267      1
                268      1
                269      1
                270      1
                271      1
                272      1
                273      1
                274      1
                275      1
                276      1
                277      1
                278      1
                279      1
                280      1
                281      1
                282      1
                283      1
                284      1
                285      1
                286      1
                287      1
                288      1
                289      1
                290      1
                291      1
                292      1
                293      1
                294      1
                295      1
                296      1
                297      1
                298      1
                299      1
                300      1
                301      1
                302      1
                303      1
                304      1
                305      1
                306      1
                307      1
                308      1
                309      1
                310      1
                311      1
                312      1
                313      1
                314      1
                315      1
                316      1
                317      1
                318      1
                319      1
                320      1
                321      1
                322      1
                323      1
                324      1
                325      1
                326      1
                327      1
                328      1
                329      1
                330      1
                331      1
                332      1
                333      1
                334      1
                335      1
                336      1
                337      1
                338      1
                339      1
                340      1
                341      1
                342      1
                343      1
                344      1
                345      1
                346      1
                347      1
                348      1
                349      1
                350      1
                351      1
                352      1
                353      1
                354      1
                355      1
                356      1
                357      1
                358      1
                359      1
                360      1
                361      1
                362      1
                363      1
                364      1
                365      1
                366      1
                367      1
                368      1
                369      1
                370      1
                371      1
                372      1
                373      1
                374      1
                375      1
                376      1
                377      1
                378      1
                379      1
                380      1
                381      1
                382      1
                383      1
                384      1
                385      1
                386      1
                387      1
                388      1
                389      1
                390      1
                391      1
                392      1
                393      1
                394      1
                395      1
                396      1
                397      1
                398      1
                399      1
                400      1
                401      1
                402      1
                403      1
                404      1
                405      1
                406      1
                407      1
                408      1
                409      1
                410      1
                411      1
                412      1
                413      1
                414      1
                415      1
                416      1
                417      1
                418      1
                419      1
                420      1
                421      1
                422      1
                423      1
                424      1
                425      1
                426      1
                427      1
                428      1
                429      1
                430      1
                431      1
                432      1
                433      1
                434      1
                435      1
                436      1
                437      1
                438      1
                439      1
                440      1
                441      1
                442      1
                443      1
                444      1
                445      1
                446      1
                447      1
                448      1
                449      1
                450      1
                451      1
                452      1
                453      1
                454      1
                455      1
                456      1
                457      1
                458      1
                459      1
                460      1
                461      1
                462      1
                463      1
                464      1
                465      1
                466      1
                467      1
                468      1
                469      1
                470      1
                471      1
                472      1
                473      1
                474      1
                475      1
                476      1
                477      1
                478      1
                479      1
                480      1
                481      1
                482      1
                483      1
                484      1
                485      1
                486      1
                487      1
                488      1
                489      1
                490      1
                491      1
                492      1
                493      1
                494      1
                495      1
                496      1
                497      1
                498      1
                499      1
                500      1
                501      1
                502      1
                503      1
                504      1
                505      1
                506      1
                507      1
                508      1
                509      1
                510      1
                511      1
                512      1
                513      1
                514      1
                515      1
                516      1
                517      1
                518      1
                519      1
                520      1
                521      1
                522      1
                523      1
                524      1
                525      1
                526      1
                527      1
                528      1
                529      1
                530      1
                531      1
                532      1
                533      1
                534      1
                535      1
                536      1
                537      1
                538      1
                539      1
                540      1
                541      1
                542      1
                543      1
                544      1
                545      1
                546      1
                547      1
                548      1
                549      1
                550      1
                551      1
                552      1
                553      1
                554      1
                555      1
                556      1
                557      1
                558      1
                559      1
                560      1
                561      1
                562      1
                563      1
                564      1
                565      1
                566      1
                567      1
                568      1
                569      1
                570      1
                571      1
                572      1
                573      1
                574      1
                575      1
                576      1
                577      1
                578      1
                579      1
                580      1
                581      1
                582      1
                583      1
                584      1
                585      1
                586      1
                587      1
                588      1
                589      1
                590      1
                591      1
                592      1
                593      1
                594      1
                595      1
                596      1
                597      1
                598      1
                599      1
                600      1
                601      1
                602      1
                603      1
                604      1
                605      1
                606      1
                607      1
                608      1
                609      1
                610      1
                611      1
                612      1
                613      1
                614      1
                615      1
                616      1
                617      1
                618      1
                619      1
                620      1
                621      1
                622      1
                623      1
                624      1
                625      1
                626      1
                627      1
                628      1
                629      1
                630      1
                631      1
                632      1
                633      1
                634      1
                635      1
                636      1
                637      1
                638      1
                639      1
                640      1
                641      1
                642      1
                643      1
                644      1
                645      1
                646      1
                647      1
                648      1
                649      1
                650      1
                651      1
                652      1
                653      1
                654      1
                655      1
                656      1
                657      1
                658      1
                659      1
                660      1
                661      1
                662      1
                663      1
                664      1
                665      1
                666      1
                667      1
                668      1
                669      1
                670      1
                671      1
                672      1
                673      1
                674      1
                675      1
                676      1
                677      1
                678      1
                679      1
                680      1
                681      1
                682      1
                683      1
                684      1
                685      1
                686      1
                687      1
                688      1
                689      1
                690      1
                691      1
                692      1
                693      1
                694      1
                695      1
                696      1
                697      1
                698      1
                699      1
                700      1
                701      1
                702      1
                703      1
                704      1
                705      1
                706      1
                707      1
                708      1
                709      1
                710      1
                711      1
                712      1
                713      1
                714      1
                715      1
                716      1
                717      1
                718      1
                719      1
                720      1
                721      1
                722      1
                723      1
                724      1
                725      1
                726      1
                727      1
                728      1
                729      1
                730      1
                731      1
                732      1
                733      1
                734      1
                735      1
                736      1
                737      1
                738      1
                739      1
                740      1
                741      1
                742      1
                743      1
                744      1
                745      1
                746      1
                747      1
                748      1
                749      1
                750      1
                751      1
                752      1
                753      1
                754      1
                755      1
                756      1
                757      1
                758      1
                759      1
                760      1
                761      1
                762      1
                763      1
                764      1
                765      1
                766      1
                767      1
                768      1
                769      1
                770      1
                771      1
                772      1
                773      1
                774      1
                775      1
                776      1
                777      1
                778      1
                779      1
                780      1
                781      1
                782      1
                783      1
                784      1
                785      1
                786      1
                787      1
                788      1
                789      1
                790      1
                791      1
                792      1
                793      1
                794      1
                795      1
                796      1
                797      1
                798      1
                799      1
                800      1
                801      1
                802      1
                803      1
                804      1
                805      1
                806      1
                807      1
                808      1
                809      1
                810      1
                811      1
                812      1
                813      1
                814      1
                815      1
                816      1
                817      1
                818      1
                819      1
                820      1
                821      1
                822      1
                823      1
                824      1
                825      1
                826      1
                827      1
                828      1
                829      1
                830      1
                831      1
                832      1
                833      1
                834      1
                835      1
                836      1
                837      1
                838      1
                839      1
                840      1
                841      1
                842      1
                843      1
                844      1
                845      1
                846      1
                847      1
                848      1
                849      1
                850      1
                851      1
                852      1
                853      1
                854      1
                855      1
                856      1
                857      1
                858      1
                859      1
                860      1
                861      1
                862      1
                863      1
                864      1
                865      1
                866      1
                867      1
                868      1
                869      1
                870      1
                871      1
                872      1
                873      1
                874      1
                875      1
                876      1
                877      1
                878      1
                879      1
                880      1
                881      1
                882      1
                883      1
                884      1
                885      1
                886      1
                887      1
                888      1
                889      1
                890      1
                891      1
                892      1
                893      1
                894      1
                895      1
                896      1
                897      1
                898      1
                899      1
                900      1
                901      1
                902      1
                903      1
                904      1
                905      1
                906      1
                907      1
                908      1
                909      1
                910      1
                911      1
                912      1
                913      1
                914      1
                915      1
                916      1
                917      1
                918      1
                919      1
                920      1
                921      1
                922      1
                923      1
                924      1
                925      1
                926      1
                927      1
                928      1
                929      1
                930      1
                931      1
                932      1
                933      1
                934      1
                935      1
                936      1
                937      1
                938      1
                939      1
                940      1
                941      1
                942      1
                943      1
                944      1
                945      1
                946      1
                947      1
                948      1
                949      1
                950      1
                951      1
                952      1
                953      1
                954      1
                955      1
                956      1
                957      1
                958      1
                959      1
                960      1
                961      1
                962      1
                963      1
                964      1
                965      1
                966      1
                967      1
                968      1
                969      1
                970      1
                971      1
                972      1
                973      1
                974      1
                975      1
                976      1
                977      1
                978      1
                979      1
                980      1
                981      1
                982      1
                983      1
                984      1
                985      1
                986      1
                987      1
                988      1
                989      1
                990      1
                991      1
                992      1
                993      1
                994      1
                995      1
                996      1
                997      1
                998      1
                999      1
                1000      1

```

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:HSDRV.A96

OBJECT FILE: :F3:HSDRV.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			1	\$TITLE('HSDRV.A96: Driver module for HSO PWM program')
			2	
			3	HSDRV MODULE MAIN, STACKSIZE(8)
			4	
			5	
			6	PUBLIC HSO_ON_0, HSO_OFF_0
			7	PUBLIC HSO_ON_1, HSO_OFF_1
			8	PUBLIC HSO_TIME, HSO_COMMAND
			9	PUBLIC SP, TIMER1, IOSO
			10	
			11	\$INCLUDE(DEMO96.INC)
			12	\$nolist ; Turn listing off for include file
			60	; End of include file
			61	
	0028		62	rseg at 28H
			63	
			64	EXTRN OLD_STAT :byte
			65	
	0028		66	HSO_ON_0: dsw 1
	002A		67	HSO_OFF_0: dsw 1
	002C		68	HSO_ON_1: dsw 1
	002E		69	HSO_OFF_1: dsw 1
	0030		70	count: dsb 1
			71	
	2080		72	cseg at 2080H
			73	
			74	EXTRN wait :entry
			75	
	2080 FA		76	strt: DI
	2081 A1000118		77	LD SP, #100H
	2085 510F1500	E	78	ANDB OLD_STAT, IOSO, #0FH
	2089 950F00	E	79	XORB OLD_STAT, #0FH
			80	
	208C 01000055		81	initial: CX, #000000H
	208C A1000122		82	LD CX, #0100H
			83	
	2090 A100101C		84	loop: LD AX, #1000H
	2094 48221C20		85	SUB BX, AX, CX
	2098 A0221C		86	LD AX, CX
			87	
			88	
			89	
			90	
			91	
			92	
			93	
			94	
			95	
			96	
			97	
			98	
			99	
			100	

270061-79

ASSEMBLY COMPLETED, NO ERROR(S) FOUND

270061-80

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F3:HSMOD.A96

OBJECT FILE: F3:HSMOD.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE('HSMOD.A96: 8096 PWM PROGRAM MODIFIED FOR DRIVER')
2 $PAGEWIDTH(130)
3
4 ; This program will provide 3 PWM outputs on HSO pins 0-2
5 ; The input parameters passed to the program are:
6
7 ; HSO_ON_N HSO on time for pin N
8 ; HSO_OFF_N HSO off time for pin N
9
10 ; Where: Times are in timer1 cycles
11 ; FOR HSO_N takes values from 0 to 3
12
13
14
15
16 NOTE: Use this file to replace the declaration section of
17 the HSO PWM program from "$INCLUDE(DEMO96.INC)" through
18 the line prior to the label "wait". Also change the last
19 branch in the program to a "RET".
20
21 RSEG
22
23 D_STAT: DSB 1
24 extrn HSO_ON_0:word, HSO_OFF_0:word
25 extrn HSO_ON_1:word, HSO_OFF_1:word
26 extrn HSO_TIME:word, HSO_COMMAND:byte
27 extrn TIMER1:word, IOSO:byte
28 extrn SP:word
29
30 public OLD_STAT
31 OLD_STAT: DSB 1
32 NEW_STAT: INC DSB 1
33
34
35 cseg
36 PUBLIC wait
37 ; Loop until HSO holding register
38 ; is empty
39
40
41 ; For operation with interrupts 'store_stat:' would be the
42 ; entry point of the routine.
43 ; Note that a DI or PUSHF might have to be added.
44

```

270061-81

```

0004      45  store_stat:
0004 510F0002      E 46      ANDB NEW_STAT, IOSO, #OFH      ; Store new status of HSO
0008 980201      R 47      CMPB OLD_STAT, NEW_STAT
0008 DFF3      48      JE      wait
000D 740201      R 49      XORB OLD_STAT, NEW_STAT
      50
0010 3E000D      E 51      check_0:
0010 300113      R 53      JBC  OLD_STAT, 0, check_1      ; Jump if OLD_STAT(0)=NEW_STAT(0)
0013 380209      R 54      JBS  NEW_STAT, 0, set_off_0
      55
0016      56  set_on_0:
0016 B13000      E 57      LDB  HSO_COMMAND, #00110000B      ; Set HSO for timer1, set pin 0
0019 44000000      E 58      ADD  HSO_TIME, TIMER1, HSO_OFF_0 ; Time to set pin = Timer1 value
001D 2007      59      BR  check_1      ; + Time for pin to be low
      60
001F      61  set_off_0:
001F B11000      E 62      LDB  HSO_COMMAND, #00010000B      ; Set HSO for timer1, clear pin 0
0022 44000000      E 63      ADD  HSO_TIME, TIMER1, HSO_ON_0 ; Time to clear pin = Timer1 value
      64
0026      65  check_1:
0026 310113      R 66      JBC  OLD_STAT, 1, check_done ; Jump if OLD_STAT(1)=NEW_STAT(1)
0029 390209      R 67      JBS  NEW_STAT, 1, set_off_1
      68
002C      69  set_on_1:
002C B13100      E 70      LDB  HSO_COMMAND, #00110001B      ; Set HSO for timer1, set pin 1
002F 44000000      E 71      ADD  HSO_TIME, TIMER1, HSO_OFF_1 ; Time to set pin = Timer1 value
0033 2007      72      BR  check_done ; + Time for pin to be low
      73
0035      74  set_off_1:
0035 B11100      E 75      LDB  HSO_COMMAND, #00010001B      ; Set HSO for timer1, clear pin 1
0038 44000000      E 76      ADD  HSO_TIME, TIMER1, HSO_ON_1 ; Time to clear pin = Timer1 value
      77
003C      78  check_done:
003C B00201      R 79      LDB  OLD_STAT, NEW_STAT ; Store current status and
      80
003F F0      81      RET ; wait for interrupt flag
      82
      83      use "BR: wait" if this routine is used with the driver
      84
0040      85  END

```

ASSEMBLY COMPLETED. NO ERROR(S) FOUND.

270061-82

END OF OBJECT

TIME

SOURCE STATEMENT

CONTENTS SPECIFIED IN IMAGEVIEW COMMAND MORE

OBJECT LIFE: 43 HOURS 00'

SOURCE LIFE: 43 HOURS 00'

SERIES: 111 MCS-85 MICRO VERSION: A1.0

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

530001-84

SOURCE FILE F3 SP A96

OBJECT FILE F3 SP OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND. NOSB

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	
		2	\$TITLE('SP A96: SERIAL PORT DEMO PROGRAM')
		3	
		4	\$INCLUDE(DEMO96.INC)
		5	\$nolist ; Turn listing off for include file
		53	; End of include file
		54	
0028		55	rseg at 28H
		56	
0028		57	CHR: dsb 1
0029		58	SPTMP: dsb 1
002A		59	TEMPO: dsb 1
002B		60	TEMP1: dsb 1
002C		61	RCV_FLAG: dsb 1
		62	
200C		63	cseg at 200CH
		64	
200C		65	DCW ser_port_int
		66	
2080		67	cseg at 2080H
		68	
2080	A1000118	69	LD SP, #100H
		70	
2084	B12016	71	LDB IOC1, #00100000B ; Set P2.0 to TXD
		72	
		73	; Baud_rate = input frequency / (64*baud_val)
		74	; baud_val = (input frequency/64) / baud_rate
		75	
		76	
0027		77	baud_val equ 39 ; 39 = (12,000,000/64)/4800 baud
		78	
0080		79	BAUD_HIGH equ ((baud_val-1)/256) OR 80H ; Set MSB to 1
0026		80	BAUD_LOW equ (baud_val-1) MOD 256
		81	
		82	
2087	B1260E	83	LDB BAUD_REG, #BAUD_LOW
208A	B1800E	84	LDB BAUD_REG, #BAUD_HIGH
		85	
		86	
		87	
		88	
		89	
		90	
		91	
		92	
		93	
		94	
		95	
		96	
		97	
		98	
		99	
		100	

270061-83

intel

AP-248

1971

```

208D B14911      86      LDB      SPCON, #01001001B      ; Enable receiver, Mode 1
208E              87
208F              88
2090              89
2091              90
2092              91
2093 B1202A      92      STB      SBUF, CHR      ; Clear serial Port
2094              93      LDB      TEMPO, #00100000B      ; Set TI-temp
2095              94
2096 B1400B      95      LDB      INT_MASK, #01000000B      ; Enable Serial Port Interrupt
2097 FB           96      EI
2098 27FE          97      loop: BR      loop      ; Wait for serial port interrupt
2099              98
209A              99      ser_port_int:
209B              100     PUSHF
209C F2           101     rd_again:
209D              102     LDB      SPTEMP, SPSTAT      ; This section of code can be replaced
209E B01129        103     ORB      TEMPO, SPTEMP      ; with "ORB TEMPO, SP_STAT" when the
209F 90292A        104     ANDB     SPTEMP, #01100000B      ; serial port TI and RI bugs are fixed
20A0 716029        105     JNE      rd_again      ; Repeat until TI and RI are properly cleared
20A1 D7F5          106
20A2              107     get_byte:
20A3 B1501F        108     JBC      TEMPO, 6, put_byte      ; If RI-temp is not set
20A4 362A09        109     STB      SBUF, CHR      ; Store byte
20A5 C42807        110     ANDB     TEMPO, #10111111B      ; CLR RI-temp
20A6 71BF2A        111     LDB      RCV_FLAG, #0FFH      ; Set bit-received flag
20A7 B1FF2C        112
20A8              113     put_byte:
20A9 302C1B        114     JBC      RCV_FLAG, 0, continue      ; If receive flag is cleared
20AA 352A15        115     JBC      TEMPO, 5, continue      ; If TI was not set
20AB B02807        116     LDB      SBUF, CHR      ; Send byte
20AC 71DF2A        117     ANDB     TEMPO, #11011111B      ; CLR TI-temp
20AD              118
20AE 717F2B        119     ANDB     CHR, #01111111B      ; This section of code appends
20AF 990D2B        120     CMPB     CHR, #0DH      ; an LF after a CR is sent
20B0 D705          121     JNE      clr_rcv
20B1 B10A2B        122     LDB      CHR, #0AH
20B2 2002          123     BR      continue
20B3              124
20B4              125     clr_rcv:
20B5 112C          126     CLRB     RCV_FLAG      ; Clear bit-received flag
20B6              127
20B7              128     continue:
20B8 F3           129     POPF
20B9 F0           130     RET
20BA              131
20BB              132     END
20BC

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

270061-84

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F3:ATOD.A96

OBJECT FILE: F3:ATOD.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			1	*TITLE('ATOD.A96: SCANNING THE A TO D CHANNELS')
			2	
			3	*INCLUDE(DEMO96.INC)
			4	*nolist ; Turn listing off for include file
			5	; End of include file
	002B		54	RSEG at 2BH
			55	
	0020		56	BL EQU BX:BYTE
	001E		57	DL EQU DX:BYTE
			58	
	002B		59	RESULT_TABLE:
	002B		60	RESULT_1: dsw 1
	002A		61	RESULT_2: dsw 1
	002C		62	RESULT_3: dsw 1
	002E		63	RESULT_4: dsw 1
			64	
			65	
	2080		66	cseg at 2080H
			67	
			68	
	2080 A100011B		69	start: LD SP, #100H ; Set Stack Pointer
	2084 0120		70	CLR BX
			71	
	2086 550B2002		72	next: ADDB AD_COMMAND, BL, #1000B ; Start conversion on channel
			73	; indicated by BL register
			74	
	208A FD		75	NOP ; Wait for conversion to start
	208B FD		76	NOP
	208C 3B02FD		77	check: JBS AD_RESULT_LO, 3, check ; Wait while A to D is busy
			78	
	208F B0021C		79	LDB AL, AD_RESULT_LO ; Load low order result
	2092 B0031D		80	LDB AH, AD_RESULT_HI ; Load high order result
			81	
	2095 5420201E		82	ADDB DL, BL, BL ; DL=BL*2
	2099 AC1E1E		83	LDBZ DX, DL
	209C C31E2B1C		84	ST AX, RESULT_TABLE[DX] ; Store result indexed by BL*2
			85	
	20A0 1720		86	INCB BL ; Increment BL modulo 4
			87	
			88	
			89	
			90	
			91	
			92	
			93	
			94	
			95	
			96	
			97	
			98	
			99	
			100	

270061-85

APPENDIX B HSO AND A TO D UNDER INTERRUPT CONTROL

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F3:A2DHSO.A96

OBJECT FILE: F3:A2DHSO.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	\$TITLE ('A2DHSO.A96: GENERATING PWM OUTPUTS FROM A TO D INPUTS')
		2	
		3	; This program will provide 3 PWM outputs on HSO pins 0-2
		4	; and one on the PWM.
		5	
		6	; The PWM values are determined by the input to the A/D converter.
		7	
		8	;
		9	;
		10	\$INCLUDE(DEMO96.INC)
		11	\$nolist ; Turn listing off for include file
		59	; End of include file
		60	
	002B	61	RSEG AT 2BH
		62	
	001E	63	DL EQU DX:BYTE
		64	
	002B	65	ON_TIME:
	002B	66	PWM_TIME_1: DSW 1
	002A	67	HSO_ON_0: DSW 1
	002C	68	HSO_ON_1: DSW 1
	002E	69	HSO_ON_2: DSW 1
		70	
	0030	71	RESULT_TABLE:
	0030	72	RESULT_0: DSW 1
	0032	73	RESULT_1: DSW 1
	0034	74	RESULT_2: DSW 1
	0036	75	RESULT_3: DSW 1
		76	
	003B	77	NXT_ON_T: DSW 1
	003A	78	NXT_OFF_0: DSW 1
	003C	79	NXT_OFF_1: DSW 1
	003E	80	NXT_OFF_2: DSW 1
	0040	81	COUNT: DSL 1
	0044	82	AD_NUM: DSW 1 Channel being converted
	0046	83	TMP: DSW 1
	004B	84	HSO_PER: DSW 1
	004A	85	LAST_LOAD: DSB 1
		86	
	0009	87	DCM HSO = 0.01
	0004	88	DCM HSO = 0.01
	0005	89	DCM HSO = 0.01
	0000	90	DCM HSO = 0.01
	0000	91	DCM HSO = 0.01

270061-87

```

2000      87 cseg AT 2000H
2000 8020      88
2002 1D21      89 DCW start ; Timer_ovf_int
2004 8020      90 DCW Atod_done_int
2006 CC20      91 DCW start ; HSI_data_int
2006 CC20      92 DCW HSO_exec_int
2006 CC20      93
2006 CC20      94 $EJECT
2080      95
2080      96 cseg AT 2080H
2080 A100011B  97
2084 011C      98 start: LD SP, #100H ; Set Stack Pointer
2086 051C      99 CLR AX
2088 D7FC      100 wait: DEC AX ; wait approx. 0.2 seconds for
2088 D7FC      101 JNE wait ; SBE to finish communications
208A 1144      102
208A 1144      103 CLRB AD_NUM
208C A180002B  104
2090 A100014B  105 LD PWM_TIME_1, #0B0H
2094 A140002A  106 LD HSO_PER, #100H
2098 A180002C  107 LD HSO_ON_0, #040H
209C A1C0002E  108 LD HSO_ON_1, #0B0H
209C A1C0002E  109 LD HSO_ON_2, #0C0H
20A0 4500010A3B 110
20A0 4500010A3B 111 ADD NXT_ON_T, Timer1, #100H
20A5 B13606      112
20AB A03B04      113 LDB HSO_COMMAND, #00110110B ; Set HSO for timer1, set pin 0,1
20AB FD          114 LD HSO_TIME, NXT_ON_T ; with interrupt
20AC FD          115 NOP
20AD B12206      116 NOP
20B0 643B04      117 LDB HSO_COMMAND, #00100010B ; Set HSO for timer1, set pin 2
20B0 643B04      118 ADD HSO_TIME, NXT_ON_T ; without interrupt
20B3 91074A      119
20B6 B10A08      120 ORB LAST_LOAD, #00000111B ; Last loaded value was set all pins
20B9 B10A09      121 LDB INT_MASK, #00001010B ; Enable HSO and A/D interrupts
20BC FB          122 LDB INT_PENDING, #00001010B ; Fake an A/D and HSO interrupt
20BD 91010F      123 EI
20C0 65010040    124
20C4 A40042      125 loop: ORB Port1, #00000001B ; set P1.0
20C7 71FE0F      126 ADD COUNT, #01
20CA 27F1        127 ADDC COUNT+2, zero
20C7 71FE0F      128 ANDB Port1, #11111110B ; clear P1.0
20CA 27F1        129 BR loop
20C7 71FE0F      130
20CA 27F1        131 $EJECT

```

270061-88


```

132
133 .....
134 ..... HSO EXECUTED INTERRUPT .....
135 .....
136
137 HSO_exec_int:
138     PUSHF
139     ORB     Port1, #00000010B      ; Set p1.1
140
141     SUB     TMP, TIMER1, NXT_ON_T
142     CMP     TMP, ZERO
143     JLT     set_off_times
144
145 set_on_times:
146     ADD     NXT_ON_T, HSO_PER
147     LDB     HSO_COMMAND, #00110110B ; Set HSO for timer1, set pin 0.1
148     LD      HSO_TIME, NXT_ON_T
149     NOP
150     NOP
151     LDB     HSO_COMMAND, #00100010B ; Set HSO for timer1, set pin 2
152     LD      HSO_TIME, NXT_ON_T
153
154     ORB     LAST_LOAD, #00000111B ; Last loaded value was all ones
155
156     LDB     PWM_CONTROL, PWM_TIME_1 ; Now is as good a time as any
157                                           ; to update the PWM reg
158     BR      check_done
159
160
161 set_off_times:
162     JBC     LAST_LOAD, 0, check_done
163
164     ADD     NXT_OFF_0, NXT_ON_T, HSO_ON_0
165     LDB     HSO_COMMAND, #00010000B ; Set HSO for timer1, clear pin 0
166     LD      HSO_TIME, NXT_OFF_0
167
168     NOP
169     ADD     NXT_OFF_1, NXT_ON_T, HSO_ON_1
170     LDB     HSO_COMMAND, #00010001B ; Set HSO for timer1, clear pin 1
171     LD      HSO_TIME, NXT_OFF_1
172
173     NOP
174     ADD     NXT_OFF_2, NXT_ON_T, HSO_ON_2
175     LDB     HSO_COMMAND, #00010010B ; Set HSO for timer1, clear pin 2
176     LD      HSO_TIME, NXT_OFF_2
177
178     ANDB    LAST_LOAD, #11111000B ; Last loaded value was all 0s
179
180 check_done:
181     ANDB    Port1, #11111010B      ; Clear P1.1

```

270061-89

```

211B F3
211C F0
181 POPF
182 RET
183
184
185 *EJECT
186
187
188 A TO D COMPLETE INTERRUPT
189
190
191 ATOD_done_int:
192 PUSHF
193 ORB Port1, #00000100B ; Set P1.2
194
195 ANDB AL, AD_RESULT_LO, #11000000B ; Load low order result
196 LDB AH, AD_RESULT_HI ; Load high order result
197 ADDB DL, AD_NUM, AD_NUM ; DL= AD_NUM #2
198 LDBZ DX, DL
199 ST AX, RESULT_TABLE[DX] ; Store result indexed by DX
200
201 CMPEB AL, #01000000B
202 JNH no_rnd ; Round up if needed
203 CMPEB AH, #0FFH ; Don't increment if AH=0FFH
204 JE no_rnd
205 INCB AH
206
207 no_rnd: LDB AL, AH ; Align byte and change to word
208 CLRB AH
209 ST AX, ON_TIME[DX]
210
211 INCB AD_NUM
212 ANDB AD_NUM, #03H ; Keep AD_NUM between 0 and 3
213
214 next: ADDB AD_COMMAND, AD_NUM, #1000B ; Start conversion on channel
215 ; indicated by AD_NUM register
216 ANDB Port1, #11111011B ; Clear P1.2
217 POPF
218 RET
219
220
221 END
2156

```

ASSEMBLY COMPLETED. NO ERROR(S) FOUND.

510061-88

270061-90

270061-91

```

2080          88      cseg at 2080h
2080          89      ;
2080          90      reset_loc:
2080          91      ; The 8096 starts executing here on reset, the program will initialize the
2080          92      ; the software serial port and run a simple test to exercise it.
2080          93      ;
2080  FA          94      di
2081  A1F0001B    95      ld      sp,#0F0h
2085  C9C012     96      push   #4800
208B  EF0000     97      call    setup_serial_port
208B  B16C0B     98      ldb     int_mask,#01101100b ; serial, swt,hso,hsi
208E  FB        99      ei
208F          100     ;
208F          101     ;
208F          102     test1:
208F          103     ; A simple test of the serial port routines.
208F          104     ; While no characters are received an incrementing pattern is sent to the
208F          105     ; serial output. When a character is received the incrementing pattern
208F          106     ; "jumps" to the character received and proceeds from there.
208F          107     ;
208F          108     CR      equ      ODH ; Carriage return
208F  B10DOC     R 109     ldb     char,#CR
2092          110     testloop:
2092          111     ldbz    ax,char
2095  C81C       R 112     push   ax
2097  EF3000     R 113     call    char_out
2097          114     ;
209A  990DOC     R 115     cmpb   char,#CR ; Pause on Carriage return
209D  D706       116     bne     nopause
209F  011C       117     clr     ax
20A1          118     pause:
20A1  071C       119     inc     ax
20A3  D7FC       120     bne     pause
20A5          121     nopause:
20A5          122     ;
20A5  170C       R 123     incb   char
20A7          124     test2:
20A7          125     call    csts ; char ready?
20AA  98001C     R 126     cmpb   al,0
20AD  DFE3       127     be      testloop ; loop if not
20AF  EF4C00     R 128     call    char_in
20B2  B01C0C     R 129     ldb     char,al
20B5  27DB       130     br      testloop
20B5          131     $eject

```

```

END LOC 037EC4
SOURCE LIFE: 13 08081 007
SOURCE LIFE: 13 08081 007
SERIES-111 MCR-89 MCR80 APPENDIX A1 C

```

270061-92


```

0000      132
0000      133      cseg
0000      134
0000      135      setup_serial_port
0000      136      ; Called on system reset to initiate the software serial port.
0000      137
0000 CC22      138      pop      cx      ; the return address
0002 CC20      139      pop      bx      ; the baud rate (in decimal)
0004 A107001E  140      ld      dx,#0007h      ; dx:ax:=500,000 (assumes 12 Mhz crystal)
0008 A120A11C  141      ld      ax,#0A120h
000C 8C201C      142      divu     ax,bx      ; calculate the baud count (500,000/baudrate)
000F C0081C      143      st      ax,baud_count
0012 C00600      144      st      0,serial_out      ; clear serial out
0015 B16016      145      ldb      ioc1,#01100000b      ; Enable HSD, 5 and Txd
0018 3E15FD      146      bbs      ios0,6,$      ; Wait for room in the HSD CAM
0018      147      ; and issue a MARK command.
0018 44140A0A  R  148      add      txd_time,timer1,20
001F B13506      149      ldb      hso_command,#mark_command
0022 A00A04      150      ld      hso_time,txd_time
0025 1102      R  151      clrb     rcve_buf      ; clear out the receive variables
0027 1103      R  152      clrb     rcve_reg
0029 1101      R  153      clrb     rcve_state
002B EF4800      154      call    init_receive      ; setup to detect a start bit
002E E322      155      br      [cx]      ; return
002E      156      $eject
002E      157
0030      158      char_out
0030      159      ; Output character to the software serial port
0030      160
0030 CC22      161      pop      cx      ; the return address
0032 CC20      162      pop      bx      ; the character for output
0034 B10121      163      ldb      (bx+1),#01h      ; add the start and stop bits
0037 642020      164      add      bx,bx      ; to the char and leave as 16 bit
003A      165      wait_for_xmit:
003A 880006      R  166      cmp      serial_out,0      ; wait for serial_out=0 (it will be cleared by
003D D7FB      167      bne     wait_for_xmit      ; the hso interrupt process)
003F C00620      R  168      st      bx,serial_out      ; put the formatted character in serial_out
0042 E322      169      br      [cx]      ; return to caller
0042      170
0044      171      csts:
0044      172      ; Returns "true" (ax<>0) if char_in has a character.
0044      173
0044 011C      174      clr      ax
0046 300102      R  175      bbc      rcve_state,0,csts_exit
0049 071C      176      inc      ax
004B      177      csts_exit:
004B F0      178      ret
004C      179
004C      180      char_in:

```

270061-93

```

181 ; Get a character from the software serial port
182 ;
183 ; wait for character ready
184 bbc rcve_state,0,char_in
185 pushf ; set up a critical region
186 andb rcve_state,#not(rxrdy)
187 ldbrie al,rcve_buf
188 popf ; leave the critical region
189 ret
190 $eject
191
192 hso_isr:
193 ; Fields the hso interrupts and performs the serialization of the data.
194 ; Note: this routine would be incorporated into the hso service strategy
195 ; for an actual system.
196
197 cseg at 2006h
198 dcm hso_isr ; Set up vector
199
200 cseg
201 pushf
202 add txd_time,baud_count
203 cmp serial_out,0 ; if character is done send a mark
204 be send_mark
205 shr serial_out,#1 ; else send bit 0 of serial_out and shift
206 bc send_mark ; serial_out left one place.
207
208 send_space:
209 ldb hso_command,#space_command
210 ld hso_time,txd_time
211 br hso_isr_exit
212
213 send_mark:
214 ldb hso_command,#mark_command
215 ld hso_time,txd_time
216
217 hso_isr_exit:
218 popf
219 ret
220 $eject
221
222 init_receive:
223 ; Called to prepare the serial input process to find the leading edge of
224 ; a start bit.
225
226 ldb loc0,#00000000b ; disconnect change detector
227 ldb hsi_mode,#00100000b ; negative edges on HSI 2
228
229 flush_fifo:
230 orb ios1_save,ios1
231 bbs ios1_save,7,flush_fifo_done
232 ldb al,hsi_status
233 ld ax,hsi_time ; trash the fifo entry

```

```

0088 717F00      R    231      andb    ios1_save,#not(80h)      ; clear bit 7.
008B 27EF        232      br        flush_fifo
008D            233      flush_fifo_done:
008D B11015      234      ldb      ioc0,#00010000b      ; connect HSI.2 to detector
0090 F0          235      ret
                236
                237
                238
0091            239      hsi_isr:
                240      ; Fields interrupts from the HSI unit, used to detect the leading edge
                241      ; of the START bit
                242      ; Note: this routine would be incorporated into the HSI strategy of an actual
                243      ; system.
                244
2004            245      cseg at 2004h
2004 9100      R    246      dcw      hsi_isr      ; setup the interrupt vector
                247
0091            248      cseg
0091 F2 ED      249      pushf
0092 C81C      250      push    ax
0094 B0061C      251      ldb      al,hsi_status
0097 A00404      R    252      ld      sample_time,hsi_time
009A 341C15      253      bbc      al,4,exit_hsi
009D 3F15FD      254      bbs      ios0,7,$      ; wait for room in HSO holding reg
00A0 A0081C      R    255      ld      ax,baud_count      ; send out sample command in 1/2
00A3 08011C      256      shr      ax,#1      ; bit time
00A6 641C04      R    257      add     sample_time,ax
00A9 B11806      258      ldb      hso_command,#sample_command
00AC C00404      R    259      st      sample_time,hso_time
00AF B10015      260      ldb      ioc0,#00000000b      ; disconnect hsi.2 from change detector
00B2 00A 170201  261      exit_hsi:
00B2 CC1C      262      pop     ax
00B4 F3 800705  263      popf
00B5 F0 30091D  264      ret
                265      $reject
                266
00B6 005 5010  267      software_timer_isr:
                268      ; Fields the software timer interrupt, used to deserialize the incoming data.
                269      ; Note: this routine would be incorporated into the software timer strategy
                270      ; in an actual system.
                271
200A 005 180703  272      cseg at 200ah
200A B60036010E  R    273      dcw      software_timer_isr      ; setup vector
                274
00B6            275      cseg
00B6 F2 5031  276      pushf
00B7 901600      R    277      orb      ios1_save,ios1_save
00BA 71FE00      R    278      andb    ios1_save,#not(01h)      ; clear bit 0
00BD 51FC0100      R    279      andb    0,rcv_state,#0Fch      ; All bits except rxrdy and overrun=0
00C1 D70C      280      bne     process_data
                281

```

```

00C3          281  process_start_bit:
00C3 350604    282      bbc     hsi_status,5,start_ok
00C6 2FAE      283      call    init_receive
00C8 2032      284      br      software_timer_exit
00CA          285  start_ok:
00CA 910401    286      orb     rcve_state,#rip ; set receive in progress flag
00CD 2021      287      br      schedule_sample
00CF          288
00CF          289  process_data:
00CF 3F010E    290      bbs     rcve_state,7,check_stopbit
00D2 180103    291      shr     rcve_reg,#1
00D5 350603    292      bbc     hsi_status,5,datazero
00DB 918003    293      orb     rcve_reg,#80h ; set the new data bit
00DB          294  datazero:
00DB 751001    295      add     rcve_state,#10h ; increment bit count
00DE 2010      296      br      schedule_sample
00E0          297
00E0          298  check_stopbit:
00E0 3506FD    299      bbc     hsi_status,5,$ ; DEBUG ONLY
00E3 800302    300      ldb     rcve_buf,rcve_reg
00E6 910101    301      orb     rcve_state,#rxrdy
00E9 710301    302      and     rcve_state,#03h ; Clear all but ready and overrun bits
00EC 2F88      303      call    init_receive
00EE 200C      304      br      software_timer_exit
00F0          305
00F0          306  schedule_sample:
00F0 3F15FD    307      bbs     ios0,7,$ ; wait for holding reg empty
00F3 B11806    308      ldb     hso_command,#sample_command
00F6 640804    309      add     sample_time,baud_count
00F9 C00404    310      st      sample_time,hso_time
00FC          311
00FC          312  software_timer_exit:
00FC F3        313      pop     popf
00FD F0        314      ret
00FE          315
00FE          316
00FE          317  end

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

270061-96

002C 86 tmr2_old: dsl 1
 0030 87 position: dsl 1
 0034 88 des_pos: dsl 1
 0038 89 pos_err: dsl 1
 003C 90 delta_p: dsl 1
 0040 91 time: dsl 1
 0044 92 des_time: dsl 1
 0048 93 time_err: dsl 1
 94
 0050 95 \$EJECT
 0054 96
 004C 97 last_time_err: dsw 1
 004E 98 last_pos_err: dsw 1
 0050 99 pos_delta: dsw 1
 0052 100 time_delta: dsw 1
 0054 101 last_pos: dsw 1
 0056 102 last1_time: dsw 1
 0058 103 last2_time: dsw 1
 005A 104 boost: dsw 1
 005C 105 tmp1: dsw 1
 005E 106 out_ptr: dsw 1
 0060 107 offset: dsw 1
 0062 108 nxt_pos: dsw 1
 0064 109 rpwr: dsw 1
 0066 110 old_t2: dsw 1
 111
 0068 112 direct: dsw 1 ; 1=forward, 0=reverse
 0069 113 pwm_dir: dsw 1
 006A 114 hsi_s0: dsw 1
 006B 115 last_stat: dsw 1
 006C 116 pwm_pwr: dsw 1
 006D 117 ios1_bak: dsw 1
 006E 118 TR_COL: dsw 1 ; COLLECT TRACE IF TR_COL=00
 006F 119 main_dly: dsw 1
 120
 0070 121 max_pwr: dsw 1
 0072 122 max_brk: dsw 1
 0074 123 max_hold: dsw 1
 0076 124 vel_pnt: dsw 1
 0078 125 brk_pnt: dsw 1
 007A 126 pos_pnt: dsw 1
 007C 127 HSQO_dly: dsw 1
 007E 128 swt1_dly: dsw 1
 0080 129 swt2_dly: dsw 1
 0082 130 min_hsi: dsw 1
 0084 131 min_hsil: dsw 1
 0086 132 max_hsil: dsw 1
 133
 0100 dseg at 100H

550081-81

Q XIDM39QA
 MARBORP JORTNOC ROTOM

270061-98

```

0100                                     136
0102 44150404 532 137 mode_view: dsb 1
0104 013609 534 138 count_out: dsb 1
5113 LD 533 139 err_view: dsb 1
5119 LD 535 140
5115 44150404 531 141
5106 013609 530 142 $eject dsb 1
510C 013608 534 143
5108 013608 530 144 PIN#0000 PORT10 FLAG USAGE
5104 0107P6 531 145
5109 80009C 539 146 22: P1.0 mode0 0 mode1 1 mode2 1 or 0
5105 01005495 532 147 23: P1.1 0 0 1 1
50LE 010001V 534 148 24: P1.2 software timer 2 routine enter/leave
50LV 010001V 533 149 25: P1.3 Main program toggle
50LF 010001V 535 150 26: P1.4 HSI overflow toggle
50LS 010001V 531 151 27: P1.5 software timer 0 routine enter/leave
50ES 010001V 530 152 28: P1.6 hsi_int enter/leave
50EE 010001V 530 153 29: P1.7 software timer 1 routine enter/leave
50EV 010001V 534 154 40: P2.6 Input direction (0=reverse, 1=forward)
50EY 010001V 530 155 45: P2.7 direction 0=rev, 1=fwd
50EY 010001V 531 156
2000 010001V 539 157 cseg at 00000000
2000 00220099 532 158 dcw timer_ovf_int
2002 10200084 534 159 dcw atod_done_int
2004 04240085 533 160 dcw hsi_data_int
2006 8022012E 535 161 dcw hso_exec_int
2008 1020 531 162 dcw hsi_0_int
200A 2022 530 163 dcw soft_tmr_int
200C 10200000 534 164 dcw ser_port_int
200E 102000 530 165 dcw external_int
2010 0100 531 166
2010 0100 530 167 atod_done_int: dsb
2010 0100 530 168 hsi_0_int: dsb
2010 0100 530 169 ser_port_int: dsb
2010 0100 530 170 external_int: dsb
2080 0100 531 171
2080 0100 530 172 cseg at 00000000
2080 0100 530 173
2080 A1F00018 534 174 init: ld sp, #0F0H
2084 B1FF17 530 175 ldb pwm_control, #0FFH
2087 116800 531 176
2089 A170175C 532 177 clrbr direct
208D 055C 534 178 ld tmp1, #6000 ; wait about 3 seconds for motor
208F E068FD 533 179 delay: dec tmp1 ; to come to a stop
2092 88005C 535 180 djnz direct, $ ; wait 0.512 milliseconds
2095 D2F612 531 181 cmp tmp1, zero
2097 B1FF0F 530 182 jgt delay
209A B1FF10 534 183 ldb port1, #0FFH
209D 0100 530 184 ldb port2, #0FFH
209E 0100 530 185
209F 0100 530 186
20A0 0100 530 187
20A1 0100 530 188
20A2 0100 530 189
20A3 0100 530 190
20A4 0100 530 191
20A5 0100 530 192
20A6 0100 530 193
20A7 0100 530 194
20A8 0100 530 195
20A9 0100 530 196
20AA 0100 530 197
20AB 0100 530 198
20AC 0100 530 199
20AD 0100 530 200

```

270061-99

```

209D B12516      186      ldb      IOC1,#00100101B ; Disable HSD 4,HSD 5, HSI_INT=first,
187              187              ; Enable PWM,TXD,TIMER1_OVRFLOW_INT
188              188
20A0 71FC0F      189      andb     Port1,#11111100BH ; clear P1.0,1 (set mode 0)
20A3 B19903      190      ldb      HSI_mode,#10011001B ; set hsi 1,3 -, hsi.0,2 +
20A6 B15715      191      ldb      IOC0,#01010111B ; Enable all hsi
20A5 B1002C      192      ldb      T2_CLOCK,T2CLK, T2RST=T2RST ; T2 CLOCK=T2CLK, T2RST=T2RST
20B1 B095ED      193      ldb      timer2 ; Clear timer2
20B0 B22C        194      $eject
20B4 B13012C     195      ldb      zero,hs1_time
20A9 A00400      196      ldb      time
20AC 0140        197      clr      time+2
20AE 0142        198      clr      timer_2+2
20B0 012B        199      clr      timer_2+2
20B2 012A        200      clr      position+2
20B4 0130        201      clr      position+2
20B6 0132        202      clr      last_pos
20B8 0154        203      clr      des_pos
20BA 0134        204      clr      des_pos+2
20BC 0136        205      clr      des_time
20BE 0144        206      clr      des_time+2
20C0 0146        207      ld      last1_time,Timer1
20C2 A00A56      208      sub     last2_time,last1_time,#800H
20C5 4900085658  209      clrb    ios1_bak
20CA 116D        210      clrb    int_pending
20CC 1109        211      ld      out_ptr,#1F0H
20CE A1F0015E    212      ld      min_hsi,#min_hsi_t
20D2 A13C0082    213      ld      min_hsi1,#min_hsi1_t
20D6 A11E0084    214      ld      max_hsi1,#max_hsi1_t
20DA A1690086    215      ld      HSD0_dly,#HSD0_dly_period
20DE A16E007C    216      ld      swt1_dly,#swt1_dly_period
20E2 A1FA007E    217      ld      swt2_dly,#(swt2_dly_period)
20E6 A1FA0080    218      ld      max_pwr,#max_power
20EA A1FF0070    219      ld      max_brk,#max_brake
20EE A1FF0072    220      ld      max_hold,#maximum_hold
20F2 A1800074    221      ld      brk_pnt,#brake_pnt
20F6 A1800478    222      ld      pos_pnt,#position_pnt
20FA A164007A    223      ld      vel_pnt,#velocity_pnt
20FE A1100076    224      ld      nxt_pos,#pos_table
2102 A1002962    225      ldb      pwm_pwr,zero
2106 B0006C      226      ldb      pwm_dir,#01h ; FORWARD
2109 B10169      227
210C B12D08      228      ldb      int_mask,#00101101B ; Enable tmr_ovf, hsi, swt, HSD0.interrupts
210F B13006      229      ldb      hso_command,#30H ; set HSD0_0
2112 447C0A04    230      add     hso_time,timer1,HSD0_dly
2116 FD          231      nop
2117 FD          232      NOP
2118 B13906      233      ldb      hso_command,#39H ; set swt_1
211B 447E0A04    234      add     hso_time,timer1,swt1_dly
211D

```



```

211F FD          236      nop
2120 FD          237      nop
2121 B13A06      238      ldb     hso_command,#3AH      , set swt_2
2124 44800A04    239      add     hso_time, timer1, swt2_dly
2128 A00A40      240      ld     time, TIMER1
212B A00C2C      241      ld     tmr2_old, timer2
212E FB          242      ei
2131 503E        243      br    main_prog
212F E7CE06      244      br    main_prog
2131 512212      245      jmp    1000'8010:1010H
213E 11E00E      246      %reject
2138 900E0E      247      jmp    1000'8010:1010H
2138 900E0E      248      jmp    1000'8010:1010H
2138 900E0E      249      jmp    1000'8010:1010H
2138 900E0E      250      jmp    1000'8010:1010H
2138 900E0E      251      jmp    1000'8010:1010H
2138 900E0E      252      jmp    1000'8010:1010H
2138 900E0E      253      jmp    1000'8010:1010H
2200            254      CSEG AT 2200H
2138 900E0E      255      jmp    1000'8010:1010H
2200 900C0E      256      timer_ovf_int:
2200 F2500E      257      pushf
2201 90166D06     258      orb     ios1_bak, IOS1
2204 356D05      259      chk_t1: jbc     ios1_bak, 5, tmr_int_done
2207 0742        260      inc     time+2
2209 71DF6D      261      andb    ios1_bak, #11011111B      , clear bit 5
220C            262      tmr_int_done:
220C F3          263      popf
220D F0          264      ret
220D F0          265      ; End of timer interrupt routine
220D F0          266
220D F0          267
220D F0          268      ;-----
220D F0          269      ;-----
220D F0          270      ;-----
220D F0          271      ;-----
220D F0          272      ;-----
220D F0          273      CSEG AT 2220H
220D F0          274      ;-----
220D F0          275      ;-----
220D F0          276      soft_tmr_int:
220D F0          277      pushf
220D F0          278      orb     ios1_bak, IOS1
220D F0          279      chk_sw0:
220D F0          280      jbc     ios1_bak, 0, chk_sw1
220D F0          281      andb    ios1_bak, #11111110B      ; Clear bit 0 - end swt0
220D F0          282      call    swt0_expired
220D F0          283      chk_sw1:
220D F0          284      jbc     ios1_bak, 1, chk_sw2
220D F0          285      andb    ios1_bak, #11111101B      ; Clear bit 1
220D F0          286      ;-----
220D F0          287      ;-----

```

270061-A1

```

2230 EFC0D3      286      call      swt1_expired
2233 31ED9D      287  chk_sw2:  mov     ios1_bak,2,chk_sw23
2233 326D06      288      jbc      ios1_bak,#11111011B
2236 71FB6D      289      andb     ios1_bak,#11111011B      ; Clear bit 2
2239 EF4401      290      call      swt2_expired
223C 31ED9D      291  chk_sw3:  mov     ios1_bak,4,swt_int_done
223C 346D03      292      jbc      ios1_bak,#11111011B
223F 71F76D      293      andb     ios1_bak,#11111011B      ; Clear bit 3
223F 71F76D      294      call      swt3_expired
2242 31ED9D      295      swt_int_done:
2242 F3          296      popf
2243 F0          297      ret      ; END OF SOFTWARE TIMER INTERRUPT ROUTINE
2243 F0          298
2243 F0          299      CSEG W1 3550H
2243 F0          300      $eject
2243 F0          301
2243 F0          302
2243 F0          303      SOFTWARE TIMER ROUTINE 0
2243 F0          304      NOW USING HSD 0 TO TRIGGER
2243 F0          305
2243 F0          306
2243 F0          307      CSEG AT 2280H
2243 F0          308      hsd_exec_int:
2243 F0          309      ; Check mode -- Update position in mode 2
2243 F0          310      push     ios1_bak
2243 F0          311      PUSHF
2243 F0          312      ldb      HSD_COMMAND,#30H
2243 F0          313      add      HSD_TIME,TIMER1,HSD0_dly
2243 F0          314
2243 F0          315      orb      port1,#00100000B      ; set P1.5
2243 F0          316      ld      Timer_2,TIMER2
2243 F0          317      jbs      Port1.1,in_mode2
2243 F0          318      CSEG W1 3540H
2243 F0          319      in_mode1:
2243 F0          320      sub      tmp1,Timer_2,old_t2      ; Check count difference in tmp1
2243 F0          321      cmp      tmp1,#2
2243 F0          322      jh      end_sw0
2243 F0          323      set_mode0:
2243 F0          324      jbc      Port1.0,end_sw0      ; if already in mode 0
2243 F0          325      andb     Port1,#11111100B      ; Clear P1.0, P1.1 (set mode 0)
2243 F0          326      ldb      IOC0,#01010101B      ; enable all HSI
2243 F0          327      ldb      last_stat,zero
2243 F0          328      br      end_sw0
2243 F0          329      CSEG W1 3530H
2243 F0          330      in_mode2:
2243 F0          331      sub      delta_p,timer_2,tmr2_old      ; get timer2 count difference
2243 F0          332      ld      tmr2_old,timer_2
2243 F0          333      jbc      direct,0,in_rev
2243 F0          334
2243 F0          335
2243 F0          336
2243 F0          337

```

270061-A2

```

22B3 643C30      336  in_fwd: add    position,delta_p
22B6 A40032      337      addc   position+2,zero
22B9 2006        338      br     chk_mode
22B8 683C30      339
22BE A80032      340  in_rev: sub    position,delta_p
22C1 4866285C     341      subc   position+2,zero
22C5 8905005C     342
22C9 D21C        343  chk_mode:
22C1 4866285C     344      sub    tmp1,Timer_2,old_t2      ; Check count difference in tmp1
22C5 8905005C     345      cmp    tmp1,#5                  ; set model if count is too low
22C9 D21C        346      jgt     end_sw0                ; count <= 5
22CB 33F661      347
22CB 71FD0F      348  set_model:
22CE 91010F      349      andb   Port1,#11111101B        ; Clear P1.1, set P1.0 (set mode 1)
22D1 B10515      350      orb    Port1,#00000001B
22D4 A00400      351      ldb    IOCO,#00000101B        ; enable HSI 0 and 1
22D7 48B40A56     352      ld     zero,HSI_TIME
22D7 48B40A56     353      sub    last1_time,Timer1,min_hsil
22D7 48B40A56     354      ; set up so (time-last2_time)>min_hsil on next HSI
22D7 48B40A56     355  $EJECT
22D7 48B40A56     356
22DB A00400      357  clr_hsi:
22DB A00400      358      ld     ZERO,HSI_TIME
22DE 717F6D      359      andb   ios1_bak,#01111111B      ; clear bit 7
22E1 90166D      360      orb    ios1_bak,ios1
22E4 3F6DF4      361      jbs    ios1_bak,7,clr_hsi      ; If hsi is triggered then clear hsi
22E7 3F6DF4      362
22E7 A02B66      363  end_sw0:
22E7 A02B66      364      ld     old_t2,TIMER_2
22EA 71DF0F      365      andb   port1,#11011111B        ; clear P1.5
22ED F3          366      POPF
22EE F0          367      ret
22EE F0          368
22EE F0          369
22EE F0          370
22EE F0          371
22EE F0          372  //////////////////////////////////////
22EE F0          373  SOFTWARE TIMER ROUTINE 2
22EE F0          374  //////////////////////////////////////
22EE F0          375  CSEG AT 2380H
22EE F0          376
22EE F0          377  swt2_expired:
22EE F0          378      pushf
22EE F0          379      ldb    hso_command,#3AH        ; set swt_2
22EE F0          380      add    hso_time,Timer1,swt2_dly
22EE F0          381
22EE F0          382      orb    port1,#00000100B        ; set port 1,2
22EE F0          383      cmp    out_ptr,#7ffH
22EE F0          384      bnh    pulsing
22EE F0          385      ld     out_ptr,#1f0H

```

270061-A3

```

386
387 pulsing:
388     jbc     tr_col,0,swt2_done
389
390     st      position+2,[out_ptr]+ ; position_high, position low
391     st      position,[out_ptr]+
392
393     st      direct,[out_ptr]+
394     st      pwm_pwr,[out_ptr]+
395
396 ; store 8 bytes externally
397
398 swt2_done:
399     sub     tmp1,timer1,last1_time
400     cmp     tmp1,#1800H
401     jnh     swt2_ret,keep,(Timer1-last1_time)<2000H
402
403     add     last1_time,#1000H
404
405 swt2_ret:
406     andb    port1,#11111011B ; clear port1.2
407     popf
408     ret
409
410 $EJECT
411
412 ; HSI DATA AVAILABLE INTERRUPT ROUTINE
413
414 ; This routine keeps track of the current time and position of the motor.
415 ; The upper word of information is provided by the timer overflow routine.
416
417 CSEG AT 2400H
418
419 now_mode_1: br in_mode_1 ; used to save execution time for
420 no_int1: br no_int ; worst case loop
421
422 hsi_data_int: pushf HSI_TIME
423     orb     port1,#01000000B ; set P1.6
424     andb    ios1_bak,#01111111B ; Clear ios1_bak.7
425     orb     ios1_bak,ios1
426     jbc     ios1_bak,7,no_int1 ; If hsi is not triggered then
427 ; jump to no_int
428
429 get_values:
430     ld      timer_2,TIMER2
431     andb    hsi_s0,HSI_STATUS,#01010101B
432     ld      time,HSI_TIME
433
434     jbs     port1,0,now_mode_1 ; jump if in mode 1
435
436 In_mode_0:
437     jbs     hsi_s0,0,a_rise

```



```

2421 3A6A2C      436      jbs      hsi_s0,2,a_fall
2424 3C6A4D      437      jbs      hsi_s0,4,b_rise
2427 3E6A5A      438      jbs      hsi_s0,6,b_fall
242A 2094        439      br       no_cnt
242C A05658      440      ;
242F A04056      441      a_rise: ld      last2_time,last1_time
2432 685840      442      ld      last1_time,time
2435 888240      443      sub     time,last2_time
2438 D906        444      cmp     time,min_hsi
243A 91010F      445      jh      tst_statr
243D B10515      446      ;set model=
2440            447      Port1,#00000001B ; Set P1.0 (in mode 1)
2440 3E6B5B      448      ldb     IOCO,#00000101B ; Enable HSI 0 and 1
2443 3C6B67      449      tst_statr:
2446 3A6B50      450      jbs     last_stat,6,going_fwd
2449 98006B      451      jbs     last_stat,4,going_rev
244C DF46        452      jbs     last_stat,2,change_dir
244E 27B2        453      cmpb    last_stat,zero
2450 A05658      454      je     first_time ; first time in mode0
2453 A04056      455      br     no_int1
2456 685840      456      ;
2459 888240      457      a_fall: ld     last2_time,last1_time
245C D906        458      ld     last1_time,time
245E 91010F      459      sub     time,last2_time
2461 B10515      460      cmp     time,min_hsi
2464 3C6B37      461      jh      tst_statf
2467 3E6B43      462      ;set model=
246A 386B2C      463      Port1,#00000001B ; Set P1.0 (in mode 1)
246D 98006B      464      ldb     IOCO,#00000101B ; Enable HSI 0 and 1
2470 DF22        465      $EJECT
2472 20570000     466      tst_statf:
2474 386B27      467      jbs     last_stat,4,going_fwd
2477 3A6B33      468      jbs     last_stat,6,going_rev
247A 3E6B1C      469      jbs     last_stat,0,change_dir
247D 98006B      470      cmpb    last_stat,zero
2480 DF12        471      je     first_time ; first time in mode0
2482 2047        472      br     no_int
2484 3A6B17      473      ;
2487 386B23      474      b_rise: jbs    last_stat,0,going_fwd
248A 3C6B0C      475      jbs     last_stat,2,going_rev
248D 98006B      476      jbs     last_stat,6,change_dir
2490 DF02        477      cmpb    last_stat,zero
2492 3031        478      je     first_time ; first time in mode0
2494 3031        479      br     no_int
2496 3031        480      ;
2498 3031        481      b_fall: jbs    last_stat,2,going_fwd
249A 3031        482      jbs     last_stat,0,going_rev
249C 3031        483      jbs     last_stat,4,change_dir
249E 3031        484      cmpb    last_stat,zero
24A0 3031        485      je     first_time ; first time in mode0
24A2 3031        486      br     no_int

```

310001-V8

270061-A5

```

2492 2037          486      br      no_int
2493 0A05          487      ;
2494 880098        488      first_time:  ;
2494 C46B6A        489      stb      hsi_s0,last_stat
2497 2072          490      br      done_chk; add delta position
2498 2072          491      ;
2499 3041          492      ;
2499 1268          493      change_dir:  ;
2499 30680F        494      notb     direct
2499 30680F        495      no_inc:  jbc     direct,0,going_rev
2499 30680F        496      ;
249E 306833        497      going_fwd:  ;
249E 914010        498      orb      PORT2,#01000000B ; set P2.6
24A1 B10168        499      ldb      direct,#01 ; direction = forward
24A4 65010030      500      add      position,#01
24AB A40032        501      addc     position+2,zero
24AB 200D          502      br      st_stat
24AD 306833        503      going_rev:  ;
24AD 71BF10        504      andb     PORT2,#0111111B ; clear P2.6
24B0 B10068        505      ldb      direct,#00 ; direction = reverse
24B3 69010030      506      sub      position,#01
24B7 AB0032        507      sbc      position+2,zero
24B8 310101        508      ;
24BA 310101        509      st_stat:  ;
24BA C46B6A        510      ldd      stb      hsi_s0,last_stat
24BD 0A05          511      load_last:  ;
24BD A0282C        512      ld      tmr2_old,timer_2
24C0 717F6D        513      no_cnt:  andb     ios1_bak,#01111111B ; clr bit 7
24C3 90166D        514      orb      ios1_bak,ios1
24C6 376D02        515      jbc      ios1_bak,7,no_int
24C9 2746          516      again:  br      get_values
24CB 71BF0F        517      ;
24CE F30098        518      no_int:  andb     port1,#0111111B ; Clear P1.6
24CF F07830        519      popf
24D0 306833        520      ret
24D0 306833        521      ;
24D0 306833        522      $EJECT  ;
24D0 306833        523      ;
24D0 306833        524      ;
24D0 306833        525      In_mode_1:  ;
24D0 306833        526      ;
24D0 51906A5C      527      andb     tmp1,hsi_s0,#01010000B
24D4 D7EA          528      jne      no_cnt
24D6 780098        529      cmp_time:  ;
24D6 780098        530      ;
24D6 A05658        531      ld      last2_time,last1_time
24D9 A04056        532      ld      last1_time,time
24DC 4858405C      533      ;
24DC 88845C        534      cmp1:  sub      tmp1,time,last2_time
24E0 88845C        535      cmp      tmp1,min_hsil
24E1 306833        536      ;

```

270061-A6

```

24E3 D914          536          jh      check_max_time
24E5              537
24E5              538      set_mode_2:
24E5 91020F        539          orb      Port1,#00000010B      ; Set P1.1 (in mode 2)
24E8 B10015        540          ldb      IOCO,#00000000B      ; Disable all HSI
24E8 A00400        541      mt_hsi: ld      zero,hsi_time      ; empty the hsi fifo
24EE 717F6D        542          andb     ios1_bak,#01111111B      ; clear bit 7
24F1 90166D        543          orb      ios1_bak,ios1
24F4 3F6DF4        544          jbs      ios1_bak,7,mt_hsi      ; If hsi is triggered then clear hsi
24F7 2012          545          br       done_chk
24F9              546
24F9 4B58405C      547      check_max_time:
24FD 88B65C        548          sub      tmp1,time,last2_time
24FD 88B65C        549          cmp      tmp1,max_hsi1      ; max_hsi = addition to min_hsi for
2500 D109          550                      ; total time
2500 D109          551          jnh      done_chk
2502              552
2502 71FC0F        553      set_mode_0:
2502 71FC0F        554          andb     Port1,#11111100B      ; clear P1.0.1 set mode 00
2503 B15515        555          ldb      IOCO,#01010101B      ; Enable all HSI
2508 80006B        556          ldb      last_stat,zero
2508 80006B        557
2508 80006B        558      done_chk:
2508 482C283C      559          sub      delta_p,timer_2,tmr2_old      ; get timer2 count difference
250F 306808        560          jbc      direct,0,add_rev
2512 0000          561      add_fwd:
2512 643C30        562          add      position,delta_p
2515 A40032        563          addc     position+2,zero
2518 27A3          564          br       load_last
251A              565      add_rev:
251A 683C30        566          sub      position,delta_p
251D AB0032        567          subc     position+2,zero
2520 279B          568          br       load_last
2520 279B          569
2520 279B          570      $eject
2520 279B          571      .....
2520 279B          572      .....      SOFTWARE TIMER ROUTINE 1      .....
2520 279B          573      .....
2520 279B          574      .....
2520 279B          575      CSEG AT 2600H
2520 279B          576
2520 279B          577      swt1_expired:
2520 279B          578
2520 279B          579          pushf
2520 279B          580          orb      port1,#10000000B      ; set port1.7
2520 279B          581
2520 279B          582          ldb      int_mask,#00001101B      ; enable HSI, Tovf, HSO
2520 279B          583
2520 279B          584          ldb      HSO_COMMAND,#39H
2520 279B          585          add      HSO_TIME,TIMER1,swt1_dly

```

270061-A7

```

260E A0464A      586
2611 A0363A      587      ld      time_err+2,des_time+2      ; Calculate time & position error
2614 4B404448    588      ld      pos_err+2,des_pos+2
261B AB424A      589      sub     time_err,des_time,time      ; values are set
261B AB424A      590      subcc  time_err+2,time+2
261B 4B30343B    591      sub     pos_err,des_pos,position
261F AB323A      592      subcc  pos_err+2,position+2
2622 FB          593
2622 FB          594      EI
2623 4B484C52    595
2627 A0484C      596      sub     time_delta,last_time_err,time_err
2627 A0484C      597      ld      last_time_err,time_err
262A 4B384E50    598
262E A0384E      599      sub     pos_delta,last_pos_err,pos_err
262E A0384E      600      ld      last_pos_err,pos_err
262E A0384E      601
262E A0384E      602      ; Time_err = Desired time to finish - current time
262E A0384E      603      ; Pos_err = Desired position to finish - current position
262E A0384E      604      ; Pos_delta = Last position error - Current position error
262E A0384E      605      ; Time_delta = Last time error - Current time error
262E A0384E      606      ; note that errors should get smaller so deltas will be
262E A0384E      607      ; positive for forward motion (time is always forward)
262E A0384E      608
262E A0384E      609
2631 88003A      610      chk_dir:
2631 88003A      611      cmp     pos_err+2,zero
2634 D60D        612      jge     go_forward
2634 D60D        613
2634 D60D        614      go_backward:
2634 D60D        615      neg     pos_err      ; Pos_err = ABS VAL (pos_err)
2634 D60D        616      ldb     pwm_dir,#00h
2634 D60D        617      cmp     pos_err+2,#0ffffH
2634 D60D        618      jne     ld_max
2634 D60D        619      br      chk_brk
2634 D60D        620
2634 D60D        621      go_forward:
2634 D60D        622      ldb     pwm_dir,#01H
2634 D60D        623      cmp     pos_err+2,zero
2634 D60D        624      je      chk_brk
2634 D60D        625      $EJECT
2634 D60D        626
2634 D60D        627      ld_max: ldb     pwm_pwr,max_pwr
2634 D60D        628      br      chk_sanity
2634 D60D        629
2634 D60D        630      Chk_brk:
2634 D60D        631      cmp     pos_err,pos_pnt      ; Position_Error now = ABS(pos_err)
2634 D60D        632      jnh     hold_position      ; position_error < position_control_point
2634 D60D        633      cmp     pos_err,brk_pnt

```

270061-A8


```

2658 D9F1          634      jh      ld_max      , position_error>brake_point
265A          635
265A 880050      636      braking:
265D D602          637      cmp      pos_delta,zero
265F 0350          638      jge      chk_delta
2661          639      neg      pos_delta
2661          640      chk_delta:
2661 887650      641      cmp      pos_delta,vel_pnt      ; velocity = pos_delta/sample_time
2664 D10D          642      jnh      hold_position      ; jmp if ABS(velocity) < vel_pnt
2666          643
2666 B0726C      644      brake:  ldb      pwm_pwr,max_brk
2669 B06824      645      ldb      tmp,direct      ; If braking apply power in opposite
266C 1224          646      notb     tmp      ; direction of current motion
266E B02469      647      ldb      pwm_dir,tmp
2671 2030          648
2671          649      br      ld_pwr
2673          650
2673          651      Hold_position:      ; position hold mode
2673 89020038      652      cmp      pos_err,#02
2677 D906          653      jh      calc_out      ; if position error < 2 then turn off power
2679 0126          654      clr      tmp+2
267B 015A          655      clr      boost
267D 201F          656      BR      output
267F          657
267F          658      calc_out:
267F 5DFF7424      659      mulub    tmp,max_hold,#255
2683 6C3824      660      mulu     tmp,pos_err      ; Tmp = pos_err * max_hold
2686 880050      661      cmp      pos_delta,zero
2689 D709          662      jne      no_bst
268B 6504005A      663      add      boost,#04      ; Boost is integral control
268F 645A26      664      add      tmp+2,boost      ; TMP+2 = MSB(pos_err*max_hold)
2692 2002          665      br      ck_max
2694 015A          666      no_bst:  clr      boost
2696 887426      667      ck_max:  cmp      tmp+2,max_hold
2699 D103          668      jnh      output
269B A07426      669      maxed:  ld      tmp+2,max_hold
269E B0266C      670      output:  ldb      pwm_pwr,tmp+2
26A1          671
26A1          672
26A1          673      chk_sanity:
26A1 2000          674      br      ld_pwr
26A3          675      ;;
26A3          676      ;;
26A3          677      $EJECT
26A3          678
26A3          679      ld_pwr:
26A3 B06C64      680      ldb      rpwr,pwm_pwr
26A6 1264          681      notb     rpwr
26A8 3B690A      682      jbs     pwm_dir,0,p2fwd
26AB          683

```

270061-A9

```

26AB FA      684 p2bkwd: DI
26AC 717F10  685 andb    port2,#01111111B      ; clear P2.7
26AF B06417  686 ldb     pwm_control,rpwr
26B2 FB      687 EI
26B3 200B    688 br      pwrset
26B5 FA      689 p2fwd: DI
26B6 918010  690 orb     port2,#10000000B      ; set P2.7
26B9 B06417  691 ldb     pwm_control,rpwr
26BC FB      692 EI
26BD 0000    693
26BD 8B004A  694 pwrset:
26CO D225    695 cmp     time_err+2,zero ; do pos_table when err is negative
26C2 89202962 696 jgt     end_p
26C6 DE06    697 br      end_p
26C8 A1002962 698
26CC 0142    699 cmp     nxt_pos,#(32+pos_table)
26CE 3005    700 jlt     get_vals ; jump if lower
26D2 0000    701 ldr     nxt_pos,#pos_table
26D4 A26334  702 clr     time+2
26D6 A26346  703 get_vals:
26D8 A26370  704
26DA A07072  705 ld      des_pos,[nxt_pos]+
26DC 646034  706 ld      des_pos+2,[nxt_pos]+
26DE A40036  707 ld      des_time+2,[nxt_pos]+
26E0 4830344E 708 ld      max_pwr,[nxt_pos]+
26E2 0000    709 ld      max_brk,max_pwr
26E4 0000    710 add     des_pos,offset
26E6 0000    711 addc    des_pos+2,zero
26E8 0000    712 sub     last_pos_err,des_pos,position
26EA F3      713
26EB FO      714 end_p: andb    port1,#01111111B      ; clear P1.7
26ED 0000    715
26EF 0000    716 popf
26F1 0000    717 ret
26F3 0000    718
26F5 0000    719 $EJECT
26F7 0000    720
26F9 0000    721
26FB 0000    722 ..... main program .....
26FD 0000    723 .....
26FF 0000    724 .....
2701 0000    725
2703 0000    726 CSEG at 2800H
2705 0000    727
2707 0000    728 MAIN_PROG:
2709 0000    729 orb     ios1_bak,ios1
270B 0000    730 jbc     ios1_bak,6,control
270D 0000    731 andb    ios1_bak,#10111111B      ; clear ios1_bak.6
270F 0000    732 orb     Port1,#00010000B      ; Compl Bit P1.4
2711 0000    733 call    HSI_DATA_INT          ; prevent lockup
2713 0000    734

```

270061-B0

```

280F          734 control:
280F 912D0B    735         orb     int_mask,#00101101B    ; enable hsi, hso, swt, tovf interrupts
2812 FD       736         nop
2813 FD       737         nop
2814 FD       738         nop
2815 E06FFD    739         djnz    main_dly,$
2818 FD       740         nop
2819 95080F    741         xorb    port1,#00001000B    ; compliment p1.3
281C 27E2     742         BR      MAIN_PROG
                743
                744
2900          745 CSEG AT 2900H
                746
2900          747 pos_table:
                748
2900 00000000    749         dcl     00000000H    ; position 0
2904 20008000    750         dcw     0020H, 0080H    ; next time, power
2908 00C00000    751         dcl     0000C000H    ; position 1
290C 40004000    752         dcw     0040H, 0040H    ; next time, power
2910 00000000    753         dcl     00000000H    ; position 2
2914 6000C000    754         dcw     0060H, 00C0H    ; next time, power
2918 0080FFFF    755         dcl     0FFFF8000H    ; position 3
291C 80008000    756         dcw     0080H, 0080H    ; next time, power
                757
2920 00080000    758         dcl     00000800H    ; position 4
2924 58008000    759         dcw     0058H, 0080H    ; next time, power
2928 00300000    760         dcl     00003000H    ; position 5
292C 7000FF00    761         dcw     0070H, 00FFH    ; next time, power
2930 00000000    762         dcl     00000000H    ; position 6
2934 9000F000    763         dcw     0090H, 00F0H    ; next time, power
2938 00000000    764         dcl     00000000H    ; position 7
293C 9100F000    765         dcw     0091H, 00F0H    ; next time, power
                766
                767
2940          768         END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

270061-B1

October 1988

An FFT Algorithm For MCS®-96 Products Including Supporting Routines and Examples

IRA HORDEN
ECO APPLICATIONS ENGINEER

AN FFT ALGORITHM FOR MCS®-96 PRODUCTS INCLUDING SUPPORTING ROUTINES AND EXAMPLES

CONTENTS

PAGE

1.0 INTRODUCTION	6-109
2.0 PROGRAM OVERVIEW	6-109
3.0 FOURIER TRANSFORMS	6-110
4.0 THE FFT ALGORITHM	6-114
5.0 USING THE FFT	6-115
6.0 BASIC PROGRAM FOR FFTS	6-118
7.0 ASM96 PROGRAM FOR FFTS	6-122
8.0 BACKGROUND CONTROL PROGRAM	6-136
9.0 ANALOG TO DIGITAL CONVERTER MODULE	6-147
10.0 DATA PLOTTING MODULE	6-158
11.0 USING THE FFT PROGRAM	6-166
12.0 APPENDIX A	6-167
13.0 APPENDIX B	6-168
BIBLIOGRAPHY	6-182

Figures

1. Timing of the FFT Program 6-110
2. Rectangular Pulse and its Fourier Transform 6-111
3. Graphical Summation of Sine Waves 6-111
4. Square Waves from Sinusoids 6-112
5. Discrete Transform of a Square Wave 6-113
6. Bin Windows 6-115
7. Waveform is a Multiple of the Window 6-116
8. Waveform is Not a Multiple of the Window 6-116
9. Effect of Hanning Window on FFT Input 6-117
10. Bin Windows after Using Hanning Input Window 6-117
11. Flowchart of Basic Program 6-119
12. Butterflies with $N = 8$ 6-122
13. FFT Output for a Square Wave Input 6-147

Listings

1. BASIC FFT Program 6-120
2. ASM96 FFT Program 6-123
3. Main Routine 6-137
4. A to D Converter Routine 6-148
5. The Plot Module 6-159

1.0 INTRODUCTION

Intel's 8096 is a 16-bit microcontroller with processing power sufficient to perform many tasks which were previously done by microprocessors or special building block computers. A new field of applications is opened by having this much power available on a single chip controller.

The 8096 can be used to increase the performance of existing designs based on 8051s or similar 8-bit controllers. In addition, it can be used for Digital Signal Processing (DSP) applications, as well as matrix manipulations and other processing oriented tasks. One of the tasks that can be performed is the calculation of a Fast Fourier Transform (FFT). The algorithm used is similar to that in many DSP and matrix manipulation applications, so while it is directly applicable to a specific set of applications, it is indirectly applicable to many more.

FFTs are most often used in determining what frequencies are present in an analog signal. By providing a tool to identify specific waveforms by their frequency components, FFTs can be used to compare signals to one another or to set patterns. This type of procedure is used in speech detection and engine knock sensors. FFTs also have uses in vision systems where they identify objects by comparing their outlines, and in radar units to detect the dopler shift created by moving objects.

This application note discusses how FFTs can be calculated using Intel's MCS[®]-96 microcontrollers. A review of fourier analysis is presented, along with the specific code required for a 64 point real FFT. Throughout this application note, it is assumed that the reader has a working knowledge of the 8096. For those without this background the following two publications will be helpful:

1986 Microcontroller Handbook

Using the 8096, AP-248

These books are listed in the bibliography, along with other good sources of information on the MCS-96 product family and on Fast Fourier Transforms.

2.0 PROGRAM OVERVIEW

This application note contains program modules which are combined to create a program which performs an FFT on an analog signal sampled by the on-board ADC (Analog to Digital Converter) of the 8097. The results of the FFT are then provided over the serial

channel to a printer or terminal which displays the results. In the applications listed in the previous section, the data from this FFT program would be used directly by another program instead of being plotted. However, the plotted results are used here to provide an example of what the FFT does. There are four program modules discussed in this application note:

FFTRUN - Runs a 64 point FFT on its data buffer. It produces 32 14-bit complex output values and 32 14-bit output magnitudes. A fast square root routine and log conversion routine are included.

A2DCON - Fills one of two buffers with analog values at a set sample rate. The sample time can be as fast as 50 microseconds using 8x9xBH components.

PLOTSP - Plots the contents of a buffer to a serially connected printer. Routines are provided for console out and hexadecimal to decimal conversion and printing.

FTMAIN - The main module which controls the other modules.

Each of the modules will be described separately. In order to better understand how the programs work together, a brief tutorial on FFTs will be presented first, followed by descriptions of the programs in the order listed above.

The final program uses 64 real data points, taken from either a table or analog input 1. Each of the data points is a 16-bit signed number. The processing takes 12.5 milliseconds when internal RAM is used as the data space. If external RAM is used, 14 milliseconds are required. Larger FFTs can be performed by slightly modifying the programs. A 256-point FFT would take approximately 65 milliseconds, and a 1024-point version would require about 300 milliseconds.

In the program presented, the analog sampling time is set for 1 sample every 100 microseconds, providing the 64 samples in 6.4 milliseconds. The sampling time can be reduced to around 60 microseconds per point by changing a variable, and less than 50 microseconds by using the 8x9xBH series of parts, since they have a 22 microsecond A to D conversion time.

The programs are set up to be run in a sequence instead of concurrently. This provides the fastest operation if the sampling speed were reduced to the minimum possible. For the fastest operation above about 80 microseconds a sample, the programs could be run concurrently, but this would require some minor modifications of the program. Figure 1 shows the timing of the program as presented.

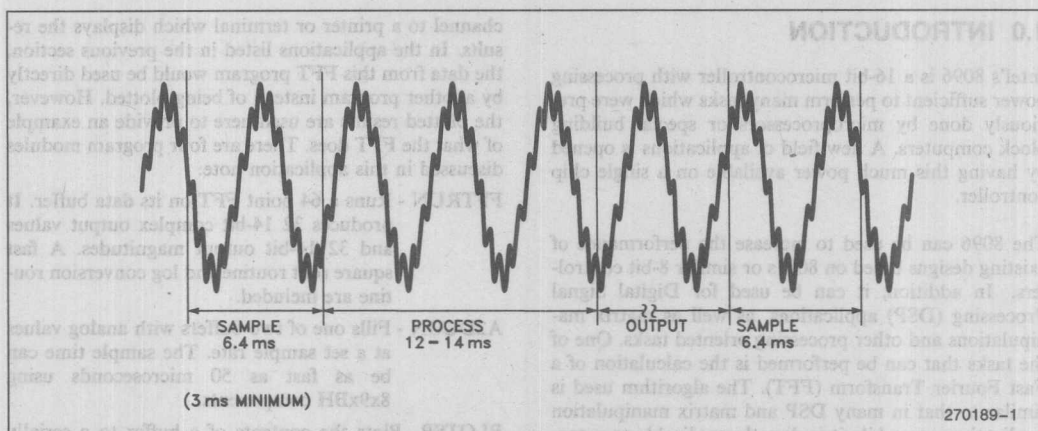


Figure 1. Timing of the FFT Program

These programs have run in the Intel Microcontroller Operation Application's Lab and produced the results presented in this application note. Since the programs have not undergone any further testing, we cannot guarantee them to be bug proof. We, therefore, recommend that they be thoroughly tested before being used for other than demonstration purposes.

3.0 FOURIER TRANSFORMS

A Fourier Transform is a useful analytical tool that is frequently ignored due to its mathematically oriented derivations. This is unfortunate, since Fourier transforms can be used without fully understanding the mathematics behind them. Of course, if one understands the theory behind these transforms, they become much more powerful.

The majority of this application note deals with how a Fast Fourier Transform (FFT) can be used for spectrum analysis. This procedure takes an input signal and separates it into its frequency components. One can almost treat the FFT as a black box, which has as its output, the frequency components and magnitudes of the input signal, much like a spectrum analyzer.

From a mathematical standpoint, Fourier Transforms change information in the time domain into the frequency domain. The theory behind the Fourier transform stems from Fourier analysis, also called frequency analysis.

There are many books on the topic of Fourier analysis, several of which are listed in the bibliography. In this application note, only the pertinent formulas and uses will be presented, not their derivations.

The main idea in Fourier analysis is that a function can be expressed as a summation of sinusoidal functions of different frequencies, phase angles, and magnitudes. This idea is represented by the Fourier Integral:

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{-j2\pi ft} dt \quad (1)$$

Where: $H(f)$ is a function of frequency
 $h(t)$ is a function of time

Since

$$e^{-j\theta} = \cos \theta - j \sin \theta \quad (2)$$

$$H(f) = \int_{-\infty}^{\infty} h(t) (\cos(2\pi ft) - j \sin(2\pi ft)) dt \quad (3)$$

Figure 2 shows a rectangular pulse and its Fourier transform. Note that the results in the frequency domain are continuous rather than discrete. The horizontal axis in Figure 2a is frequency, while that of Figure 2b is time.

In a simplified case, the varying phase angles can be removed, and the integral changed to a summation, known as a Fourier Series. All periodic functions can be described in this way. This series, as shown below, can help provide a more graphical understanding of Fourier analysis.

$$y(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(2\pi n f_0 t) + b_n \sin(2\pi n f_0 t)] \quad (4)$$

for $n = 1$ to ∞

Where $f_0 = \frac{1}{T_0}$, the fundamental frequency.

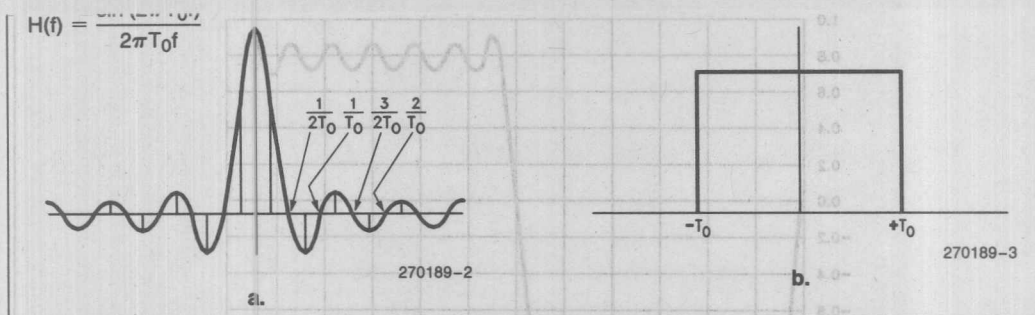


Figure 2. Rectangular Pulse and Its Fourier Transform

This formula can also be represented in complex form as:

$$\sum_{n=-\infty}^{\infty} a_n e^{j2\pi n f_0 t} \quad (5)$$

The Fourier series for a square wave is

$$\sum_{k=0}^{\infty} \frac{\sin((2k+1)2\pi f_0 t)}{(2k+1)} \quad (6)$$

If these sinusoids are summed, a square wave will be formed. Figure 3 shows the graphical summation of the first 3 terms of the series. Since the higher frequencies contribute to the squareness of the waveform at the corners, it is reasonable to compare only the flatness of the top of the waveform. The sharpness or risetime of the waveform can be determined by the highest fre-

quency term being summed. With rise and fall times of 10% of the period, the waveform generated by the first 3 terms is within 20% of ideal. At 7 terms it is within 10%, and at 20 terms it is within 5%. With a 5% risetime, it is within 20% of ideal after 5 terms, 10% after 13 terms and 5% after 32 terms. Figure 4 shows the resultant waveforms after the summation of 7, 15 and 30 terms.

Fourier analysis can be used on equation 4 to find the coefficients a_n and b_n . To make this process easier to use with a computer, a discrete form, rather than a continuous one, must be used. The discrete Fourier transform, shown in Equation 7, is a good approximation to the continuous version. The closeness of the approximation depends on several conditions which will be discussed later. The input to this transform is a set of N equally spaced samples of a waveform taken over a period of NT . The period NT is frequently referred to as the "Sampling Window".

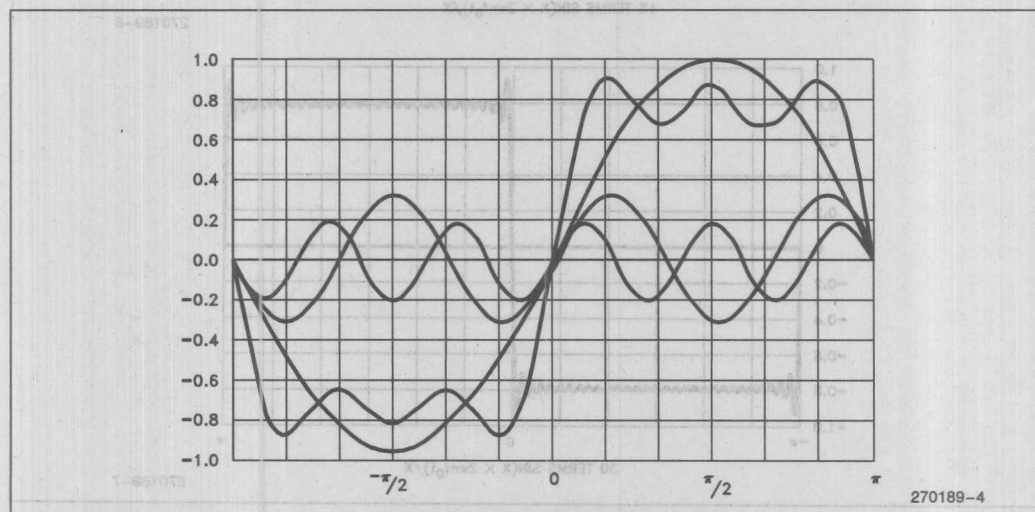


Figure 3. Graphical Summation of Sinewaves

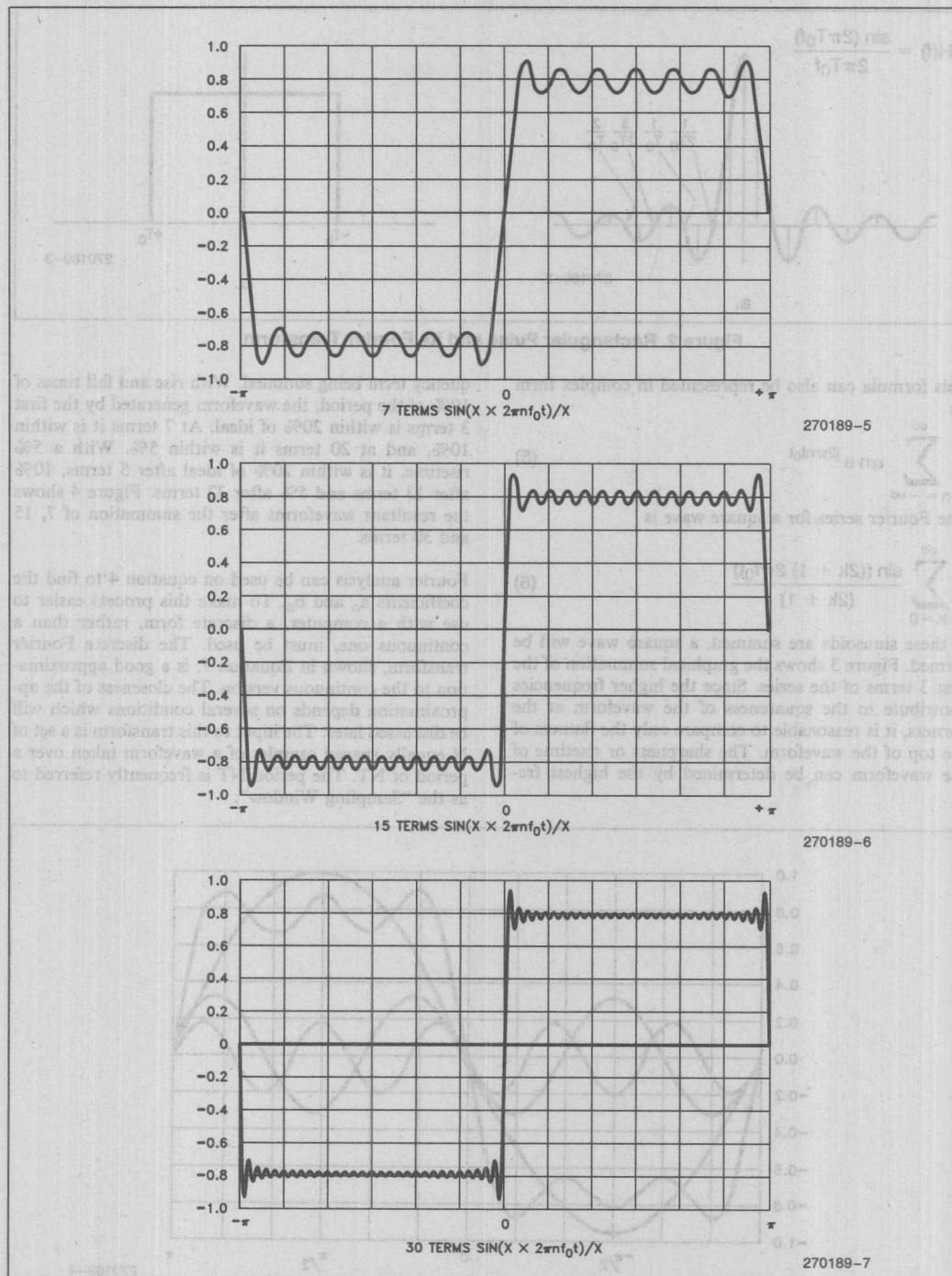


Figure 4. Square Wave from Sinusoids

$$H\left(\frac{n}{NT}\right) = \sum_{k=0}^{N-1} h(kT) e^{-j2\pi nk/N}$$

$$n = 0, 1, \dots, N-1 \quad (7)$$

Where: $H(f)$ is a function of frequency

$h(t)$ is a function of time

T is the time span between samples

N is the number of samples in the window

$n = 0, 1, 2, \dots, N-1$

This transform is used for many applications, including Fourier Harmonic Analysis. This procedure uses the transform to calculate the coefficients used in Equation 5. In order to do this, the factor T/NT must be added to the transform as follows:

$$H\left(\frac{n}{NT}\right) = \frac{T}{(NT)} \sum_{k=0}^{N-1} h(kT) e^{-j2\pi nk/N}$$

$$n = 0, 1, 2, 3, \dots, N-1 \quad (8)$$

The factor provides compensation for the number of samples taken. Note that the functions $H(f)$ and $h(t)$ are complex variables, so the simplicity of the equation can be misleading. Once the values of $h(t)$ are known, (ie.

the value of the input at the discrete times (t)), the Fourier Transform can be used to find the magnitude and phase shift of the signal at the frequencies (f).

A spectrum analyzer can provide similar information on an analog input signal by using analog filters to separate the frequency components. Regardless of its source, the information on component frequencies of a signal can be used to detect specific frequencies present in a signal or to compare one signal to another. Many lab experiments and product development tests can make use of this type of information. Using these methods, the purity of signals can be measured, specific harmonics can be detected in mechanical equipment, and noise bursts can be classified. All of this information can be obtained while still treating the FFT process as a black box.

Consider the discrete transform of a square wave as shown in Figure 5. Note that the component magnitudes, as shown in the series of Equation 6, are shown in a mirrored form in the transform. This will happen whenever only real data is used as the FFT input, if both real and imaginary data were used the output would not be guaranteed to be symmetrical. For this reason, there is duplicate information in the transform for many applications. Later in this section a method to make the most of this characteristic is discussed.

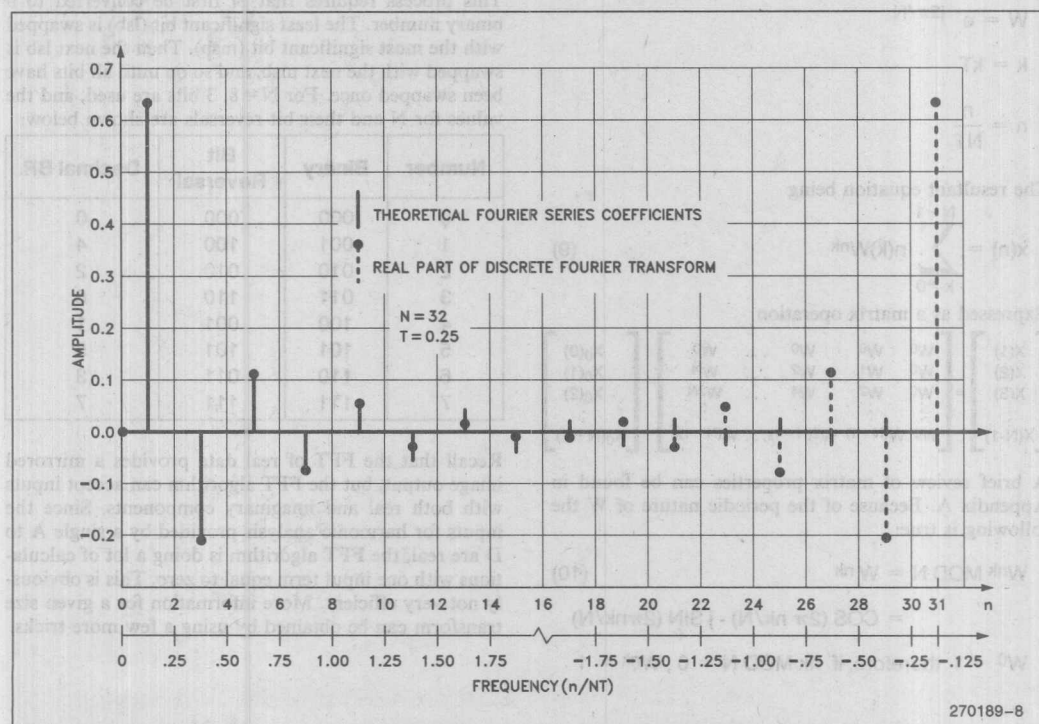


Figure 5. Discrete Transform of a Square Wave

If one looks at Equation 8, it can be seen that the calculation of a discrete Fourier transform requires N squared complex multiplications. If N is large, the calculation time can easily become unrealistic for real-time applications. For example, if a complex multiplication takes 40 microseconds, at $N = 16$, 10 milliseconds would be used for calculation, while at $N = 128$, over half a second would be needed. A Fast Fourier Transform is an algorithm which uses less multiplications, and is therefore faster. To calculate the actual time savings, it is first necessary to understand how a FFT works.

4.0 THE FFT ALGORITHM

The FFT algorithm makes use of the periodic nature of waveforms and some matrix algebra tricks to reduce the number of calculations needed for a transform. A more complete discussion of this is in Appendix A, however, the areas that need to be understood to follow the algorithm are presented here. This information need not be read if the reader's intent is to use the program and not to understand the mathematical process of the algorithm.

To simplify notation the following substitutions are made in Equation 8.

$$W = e^{-j2\pi/N}$$

$$k = kT$$

$$n = \frac{n}{NT}$$

The resultant equation being

$$x(n) = \sum_{k=0}^{N-1} n(k)Wnk \quad (9)$$

Expressed as a matrix operation

$$\begin{bmatrix} X(1) \\ X(2) \\ X(3) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & \dots & W^0 \\ W^0 & W^1 & W^2 & \dots & W^N \\ W^0 & W^2 & W^4 & \dots & W^{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W^0 & W^{(N-1)} & W^{2(N-1)} & \dots & W^{(N-1)^2} \end{bmatrix} \begin{bmatrix} X_0(0) \\ X_0(1) \\ X_0(2) \\ \vdots \\ X_0(N-1) \end{bmatrix}$$

A brief review of matrix properties can be found in Appendix A. Because of the periodic nature of W the following is true:

$$Wnk \text{ MOD } N = Wnk \quad (10)$$

$$= \cos(2\pi nk/N) - j \sin(2\pi nk/N)$$

$$W^0 = 1 \text{ therefore, if } nk \text{ MOD } N = 0, Wnk = 1$$

This reduces the calculations as several of the W terms go to 1 and the highest power of W is N . All of W values are complex, so most of the operations will have to be complex operations. We will continue to use only the W , $X(n)$ and $X_0(k)$ symbols to represent these complex quantities.

The FFT algorithm we will use requires that N be an integral power of 2. Other FFT algorithms do not have this restriction, but they are more complex to understand and develop. Additionally, for the relatively small values of N we are using this restriction should not provide much of a problem. We will define EXPONENT as log base 2 of N . Therefore,

$$N = 2^{\text{EXPONENT}}$$

The magic of the FFT, (as detailed in Appendix A), involves factoring the matrix into EXPONENT matrices, each of which has all zeros except for a 1 and a Wnk term in each row. When these matrices are multiplied together the result is the same as that of the multiplication indicated in Equation 9, except that the rows are interchanged and there are fewer non-trivial multiplications. To reorder the rows, and thus make the information useful, it is necessary to perform a procedure called "Bit Reversal".

This process requires that N first be converted to a binary number. The least significant bit (lsb) is swapped with the most significant bit (msb). Then the next lsb is swapped with the next msb, and so on until all bits have been swapped once. For $N=8$, 3 bits are used, and the values for N and their bit reversals are shown below:

Number	Binary	Bit Reversal	Decimal BR
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Recall that the FFT of real data provides a mirrored image output, but the FFT algorithm can accept inputs with both real and imaginary components. Since the inputs for harmonic analysis provided by a single A to D are real, the FFT algorithm is doing a lot of calculations with one input term equal to zero. This is obviously not very efficient. More information for a given size transform can be obtained by using a few more tricks.

It is possible to perform the FFT of two real functions at the same time by using the imaginary input values to the FFT for the second real function. There is then a post processing performed on the FFT results which separate the FFTs of the two functions. Using a similar procedure one can perform a transform on $2N$ real samples using an N complex sample transform.

The procedure involves alternating the real sample values between the real and imaginary inputs to the FFT. If, as in our example, the input to the FFT is a 2 by 32 array containing the complex values for 32 inputs, the 64 real samples would be loaded into it as follows:

N	00 01 02 03 04 05 06 07 30 31
REAL	00 02 04 06 08 10 12 14 60 62
IMAGINARY	01 03 05 07 09 11 13 15 61 63

This procedure is referred to as a pre-weave. In order to derive the desired results, the FFT is run, and then a post-weave operation is performed. The formula for the post-weave is shown below:

$$X_r(n) = \left[\frac{R(n)}{2} + \frac{R(N-n)}{2} \right] + \cos \frac{\pi n}{N} \left[\frac{I(n)}{2} + \frac{I(N-n)}{2} \right] - \sin \frac{\pi n}{N} \left[\frac{R(n)}{2} - \frac{R(N-n)}{2} \right] \quad n = 0, 1, \dots, N-1$$

$$X_i(n) = \left[\frac{I(n)}{2} - \frac{I(N-n)}{2} \right] - \sin \frac{\pi n}{N} \left[\frac{I(n)}{2} + \frac{I(N-n)}{2} \right] - \cos \frac{\pi n}{N} \left[\frac{R(n)}{2} - \frac{R(N-n)}{2} \right] \quad n = 0, 1, \dots, N-1 \quad (11)$$

Where $R(n)$ is the real FFT output value

$I(n)$ is the imaginary FFT output value

$X_r(n)$ is the real post-weave output

$X_i(n)$ is the imaginary post-weave output

Note that the output is now one-sided instead of mirrored around the center frequency as it is in Figure 5. The magnitude of the signal at each frequency is calculated by taking the square root of the sum of the squares. The magnitude can now be plotted against frequency, where the frequency steps are defined as:

$$\frac{n}{NT} \quad n = 0, 1, 2, 3, \dots, N-1$$

Where N is the number of complex samples (ie. 32 in this case) T is the time between samples

A value of zero on the frequency scale corresponds to the DC component of the waveform. Most signal analysis is done using Decibels (dB), the conversion is $\text{dB} = 10 \text{ LOG (Magnitude squared)}$. Decibels are not used as an absolute measure, instead signals are compared by the difference in decibels. If the ratio between two signals is 1:2 then there will be a 3 dB difference in their power.

5.0 USING THE FFT

There are several things to be aware of when using FFTs, but with the proper cautions, the FFT output can be used just like that of a spectrum analyzer. The

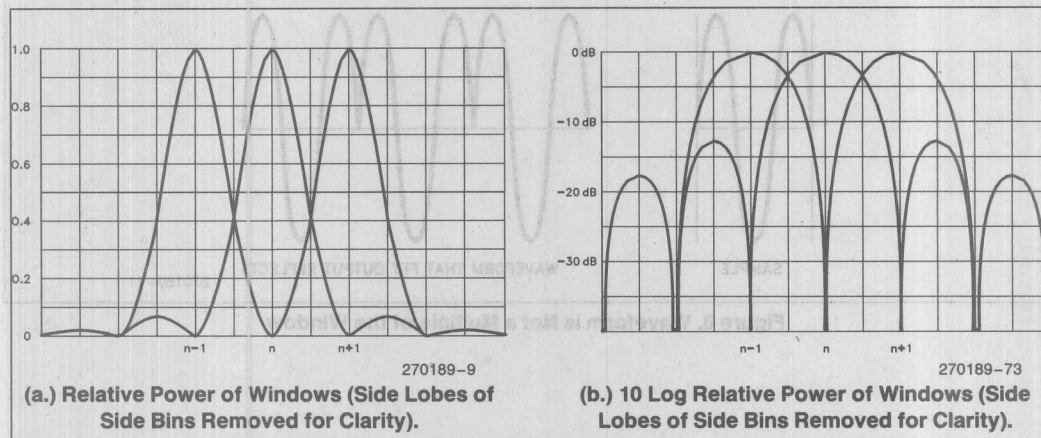


Figure 6. Bin Windows

first precaution is that the FFT is a discrete approximation to a continuous Fourier Transform, so the output will seldom fit the theoretical values exactly, but it will be very close.

Since the programs in this application note generate a one-sided transform with $N=32$, the frequency granularity is fairly coarse. Each of the frequency components output from the FFT is actually the sum of all energy within a narrow band centered on that frequency. This band of sensitivity is referred to as a "bin". The reported magnitude is the actual magnitude multiplied by the value of the bin window at the actual frequency. Figure 6 shows several bin windows. Note that these windows overlap, so that a frequency midway between the two center frequencies will be reported as energy split between both windows. Be careful not to

confuse the *sampling window* NT with *bin windows* or with the *windowing function*.

Another area of caution is the relationship of the sampling window to the frequency of the waveform. For the best accuracy, the window should cover an exact multiple of the period of the waveform being analyzed. If it covers less than one period, the results will be invalid. Other variations from ideal will not produce invalid results, just additional noise in the output.

If the sampling window does not cover an exact multiple of all of the frequency components of a waveform, the FFT results will be noisy. The reason for this is the sharp edge that the FFT sees when the edges of the window cut off the input waveform. Figure 7 shows a waveform that is an exact multiple of the window and

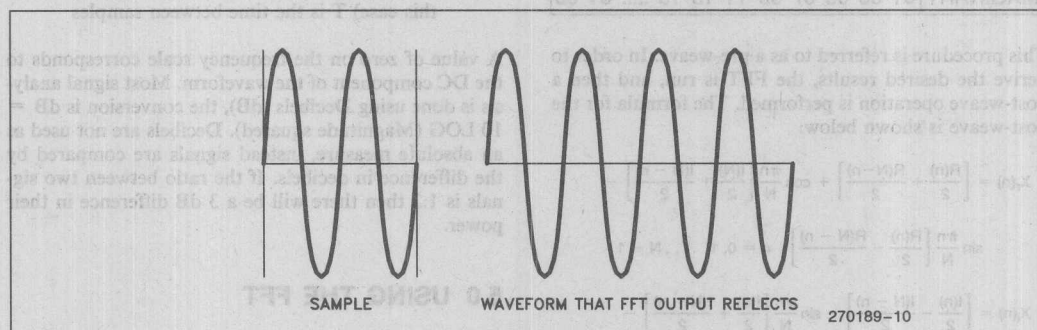


Figure 7. Waveform is a Multiple of the Window

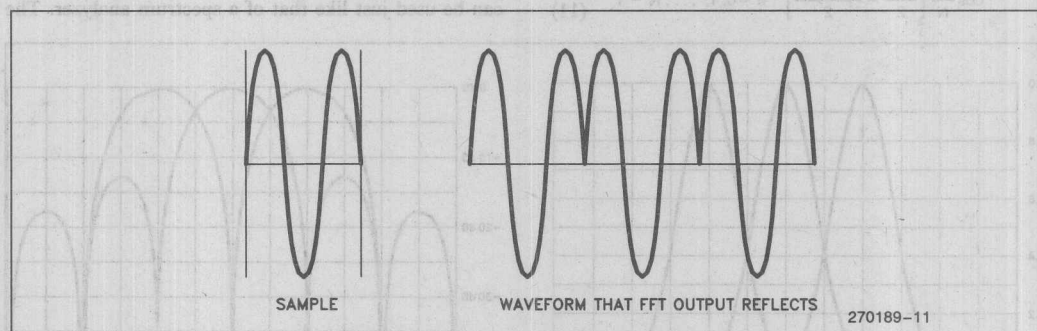


Figure 8. Waveform is Not a Multiple of the Window

the periodic waveform that the FFT output reflects. In Figure 8, the waveform is not a multiple of the window and the waveform that the FFT output reflects has discontinuities. These discontinuities contribute to the noise in an FFT output. This noise is called "spectral leakage", or simply "leakage", since it is leakage between one frequency spectrum and another which is caused by digitization of an analog process.

To reduce this leakage, a process called windowing is used. In this procedure the input data is multiplied by specific values before being used in the FFT. The term "windowing" is used because these values act as a window through which the input data passes. If the input window goes smoothly to zero at both endpoints of

the sampling window, there can be no discontinuities. Figure 9 shows a Hanning window and its effect on the input to an FFT. The Hanning window was named after its creator, Julius Von Hann, and is one of the most commonly used windows. More information on windowing and the types of windows can be found in the paper by Harris listed in the bibliography. As expected, the results of the FFT are changed because of the input windowing, but it is in a very predictable way.

Using the Hanning window results in bin windows which are wider and lower in magnitude than normal, as can be seen by comparing Figure 6 with Figure 10. For an input frequency which is equal to the center frequency of a bin window, the attenuation will be 6 dB on the center frequency. Since the bin windows are

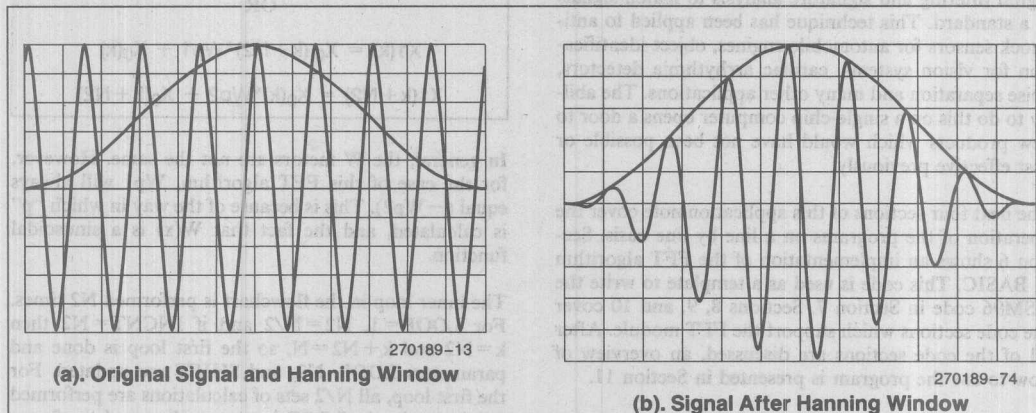


Figure 9. Effect of Hanning Window on FFT Input

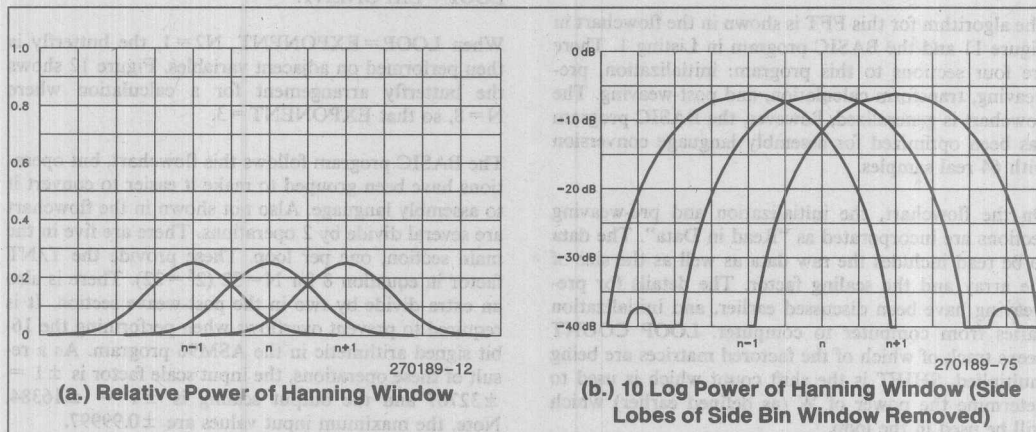


Figure 10. Bin Windows after Using Hanning Input Window

wider than normal, the input frequency will also have energy which falls into the bins on either side of center. These side bins will show a reading of 6 dB below the center window. The disadvantage of this spreading is far less than the advantage of removing leakage from the FFT output.

A set of FFT output plots are included in the Appendix. These plots show the effect of windowing on various signals. There are examples of all of the cases described above. A brief discussion of the plots is also presented.

Applications which can make use of this frequency magnitude information include a wide range of signal processing and detection tasks. Many of these tasks use digital filtering and signature analysis to match signals to a standard. This technique has been applied to anti-knock sensors for automobile engines, object identification for vision systems, cardiac arrhythmia detectors, noise separation and many other applications. The ability to do this on a single-chip computer opens a door to new products which would have not been possible or cost effective previously.

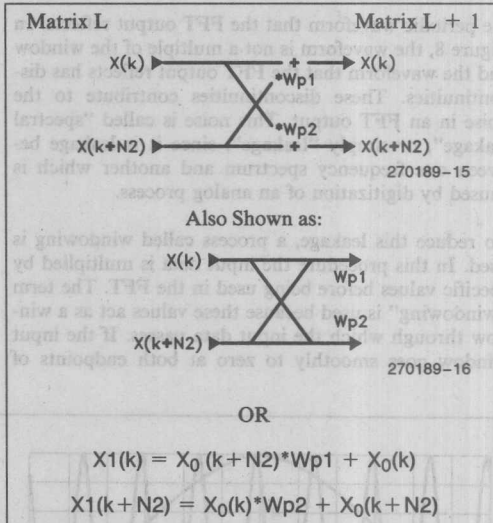
The next four sections of this application note cover the operation of the programs on a line by line basis. Section 6 shows an implementation of the FFT algorithm in BASIC. This code is used as a template to write the ASM96 code in Section 7. Sections 8, 9, and 10 cover the code sections which support the FFT module. After all of the code sections are discussed, an overview of how to use the program is presented in Section 11.

6.0 BASIC PROGRAM FOR FFTS

The algorithm for this FFT is shown in the flowchart in Figure 11 and the BASIC program in Listing 1. There are four sections to this program: initialization, pre-weaving, transform calculation, and post-weaving. The flowchart is generalized, however, the BASIC program has been optimized for assembly language conversion with 64 real samples.

On the flowchart, the initialization and pre-weaving sections are incorporated as "Read in Data". The data to be read includes the raw data as well as the size of the array and the scaling factor. The details for pre-weaving have been discussed earlier, and initialization varies from computer to computer. LOOP COUNT keeps track of which of the factored matrices are being multiplied. SHIFT is the shift count which is used to determine the power of W (as defined earlier) which will be used in the loop.

For each loop N calculations are performed in sets of two. Each calculation set is referred to as a butterfly and has the following form:



In general, the W factors are not the same. However, for the case of this FFT algorithm, Wp1 will always equal $(-Wp2)$. This is because of the way in which "p" is calculated, and the fact that W(x) is a sinusoidal function.

The inner loop in the flowchart is performed N2 times. For LOOP=1, N2=N/2 and if INCNT=N2 then $k=N2$ and $k+N2=N$, so the first loop is done and parameters LOOP, N2, and SHIFT are updated. For the first loop, all N/2 sets of calculations are performed contiguously. As LOOP increases, the number of contiguous calculations are cut in half, until LOOP=EXPONENT.

When LOOP=EXPONENT, N2=1, the butterfly is then performed on adjacent variables. Figure 12 shows the butterfly arrangement for a calculation where N=8, so that EXPONENT=3.

The BASIC program follows this flowchart, but operations have been grouped to make it easier to convert it to assembly language. Also not shown in the flowchart are several divide by 2 operations. There are five in the main section, one per loop. These provide the T/NT factor in equation 8 for $N=32$ ($2^5=32$). There is also an extra divide by two in the post-weave section. It is required to prevent overflows when performing the 16-bit signed arithmetic in the ASM96 program. As a result of these operations, the input scale factor is $\pm 1 = \pm 32767$ and the output scaling is $\pm 1 = \pm 16384$. Note, the maximum input values are ± 0.99997 .

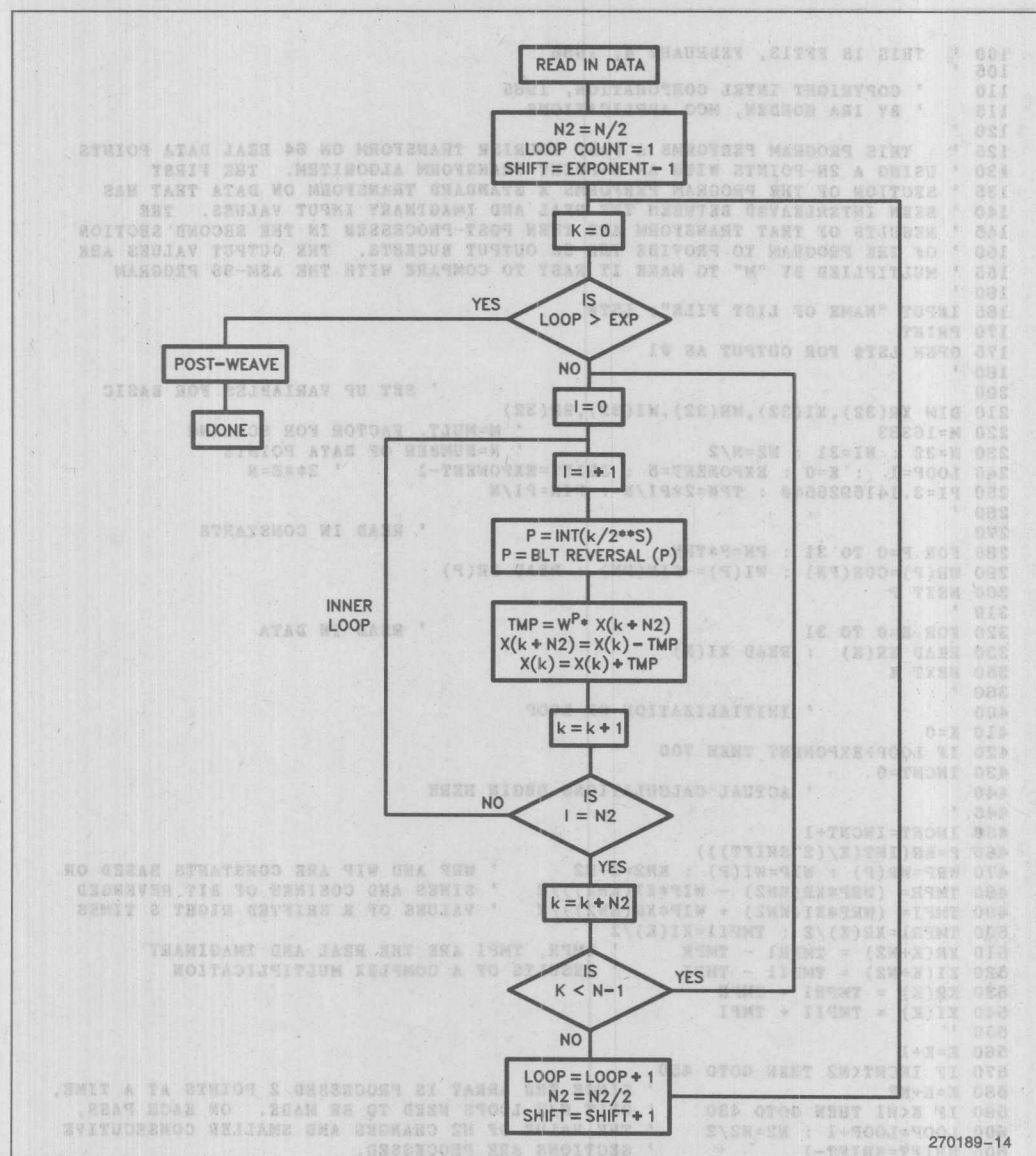


Figure 11. Flowchart of Basic Program

270189-14

```

100 ' THIS IS FFT13, FEBRUARY 4, 1986
105 '
110 ' COPYRIGHT INTEL CORPORATION, 1985
115 ' BY IRA HORDEN, MCO APPLICATIONS
120 '
125 ' THIS PROGRAM PERFORMS A FAST FOURIER TRANSFORM ON 64 REAL DATA POINTS
130 ' USING A 2N-POINTS WITH AN N-POINT TRANSFORM ALGORITHM. THE FIRST
135 ' SECTION OF THE PROGRAM PERFORMS A STANDARD TRANSFORM ON DATA THAT HAS
140 ' BEEN INTERLEAVED BETWEEN THE REAL AND IMAGINARY INPUT VALUES. THE
145 ' RESULTS OF THAT TRANSFORM ARE THEN POST-PROCESSED IN THE SECOND SECTION
150 ' OF THE PROGRAM TO PROVIDE THE 32 OUTPUT BUCKETS. THE OUTPUT VALUES ARE
155 ' MULTIPLIED BY "M" TO MAKE IT EASY TO COMPARE WITH THE ASM-96 PROGRAM
160 '
165 INPUT "NAME OF LIST FILE"; LST$
170 PRINT
175 OPEN LST$ FOR OUTPUT AS #1
180 '
200 ' SET UP VARIABLES FOR BASIC
210 DIM XR(32),XI(32),WR(32),WI(32),BR(32)
220 M=16383 ' M=MULT. FACTOR FOR SCALING
230 N=32 : N1=31 : N2=N/2 ' N=NUMBER OF DATA POINTS
240 LOOP=1 : K=0 : EXPONENT=5 : SHIFT=EXPONENT-1 ' 2**E=N
250 PI=3.141592654# : TPN=2*PI/N : PIN=PI/N
260 '
270 ' READ IN CONSTANTS
280 FOR P=0 TO 31 : PN=P*TPN
290 WR(P)=COS(PN) : WI(P)=-SIN(PN) : READ BR(P)
300 NEXT P
310 '
320 FOR K=0 TO 31 ' READ IN DATA
330 READ XR(K) : READ XI(K)
350 NEXT K
360 '
400 ' INITIALIZATION OF LOOP
410 K=0
420 IF LOOP>EXPONENT THEN 700
430 INCNT=0
440 ' ACTUAL CALCULATIONS BEGIN HERE
445 '
450 INCNT=INCNT+1
460 P=BR(INT(K/(2^SHIFT)))
470 WRP=WR(P) : WIP=WI(P) : KN2=K+N2 ' WRP AND WIP ARE CONSTANTS BASED ON
480 TMPR= (WRP*XR(KN2) - WIP*XI(KN2))/2 ' SINES AND COSINES OF BIT REVERSED
490 TMPI= (WRP*XI(KN2) + WIP*XR(KN2))/2 ' VALUES OF K SHIFTED RIGHT S TIMES
500 TMPR1=XR(K)/2 : TMPI1=XI(K)/2
510 XR(K+N2) = TMPR1 - TMPR ' TMPR, TMPI ARE THE REAL AND IMAGINARY
520 XI(K+N2) = TMPI1 - TMPI ' RESULTS OF A COMPLEX MULTIPLICATION
530 XR(K) = TMPR1 + TMPR
540 XI(K) = TMPI1 + TMPI
550 '
560 K=K+1
570 IF INCNT<N2 THEN GOTO 450
580 K=K+N2 ' SINCE THE ARRAY IS PROCESSED 2 POINTS AT A TIME,
590 IF K<N1 THEN GOTO 430 ' ONLY N/2 LOOPS NEED TO BE MADE. ON EACH PASS,
600 LOOP=LOOP+1 : N2=N2/2 ' THE VALUE OF N2 CHANGES AND SMALLER CONSECUTIVE
605 SHIFT=SHIFT-1 ' SECTIONS ARE PROCESSED.
610 GOTO 400
620 '
690 '
691 '
692 '
693 '

```

270189-17

Listing 1—BASIC FFT Program

```

694 '
695 '
696 '
697 '
700 ' POST-PROCESSING AND REORDERING BEGIN HERE
710 '
720 FOR K = 0 TO 31
730 KPIN=K*PIN
740 XRBRK=XR(BR(K)) : XIBRK=XI(BR(K)) ' CONDENSED FOR EASE OF ASM PROGRAMMING
750 XRBRNK=XR(BR(N-K)) : XIBRNX=XI(BR(N-K))
760 TI = (XIBRK+XIBRNX)/2
770 TR = (XRBRK-XRBRNK)/2
780 XRT= (XRBRK+XRBRNK)/4
790 XIT= (XIBRK-XIBRNX)/4
800 OUTR= XRT + TI*COS(KPIN)/2 - TR*SIN(KPIN)/2
810 OUTI= XIT - TI*SIN(KPIN)/2 - TR*COS(KPIN)/2
820 '
830 MAGSQ = OUTR*OUTR+OUTI*OUTI ' THE ASM-96 PROGRAM USES A TABLE LOOK-UP
840 MAG = SQR(MAGSQ) ' ROUTINE TO CALCULATE SQUARE ROOTS
845 IF MAGSQ*M < .5 THEN DECIBEL=0 : GOTO 900
847 DBFACT=M/2/32767*M ' M^2 / 64K
850 DECIBEL=10*LOG(MAGSQ*DBFACT)
860 DECIBEL=DECIBEL * .434294481#
900 GOTO 930
910 PRINT #1, USING "***** "; K,
920 PRINT #1, USING "\ " ; HEX$(M*OUTR), HEX$(M*OUTI), HEX$(M*MAG)
930 ' GOTO 950
942 PRINT #1, USING "## "; K;
943 PRINT #1, USING "###***** "; OUTR,OUTI,MAG;
945 PRINT #1, USING "###.### "; DECIBEL;
947 PRINT #1, USING "***** "; M*OUTR, M*OUTI, M*MAG
950 NEXT K
960 '
970 IF LST$<>"SCRN:" THEN PRINT #1, CHR$(12)
999 END
1000 END
1010 ' DATA FOR BR(P) - BIT REVERSAL
1020 DATA 0,16,8,24,4,20,12,28,2,18,10,26,6,22,14,30
1030 DATA 1,17,9,25,5,21,13,29,3,19,11,27,7,23,15,31
1040 ' DATA FOR XR,XI
1050 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1060 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1070 DATA -2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2
1080 DATA -2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2

```

270189-18

6

Listing 1—BASIC FFT Program (Continued)

Lines 165-175 set up the file for printing the data, this can be SCRN:, LPT1:, or any other file.

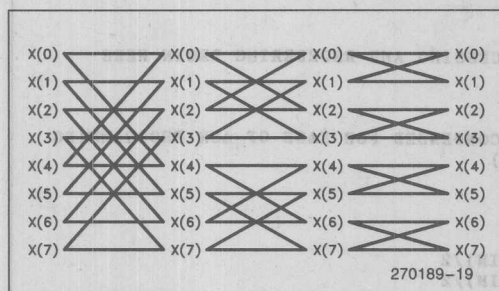


Figure 12. Butterflies with N = 8

Lines 200-310 set up the constants and calculate the WP terms which are stored in the matrices WR(p) and WI(p), for the real and imaginary component respectively.

Lines 320-350 read in the data, alternately placing it into the real and imaginary arrays. The data is scaled by 2 to make the data table simpler.

Lines 410-430 initialize the loop and test for completion.

Lines 450-620 perform the FFT algorithm. Note that all calculations are complex, with the suffixes "R" and "I" indicating real and imaginary components respectively.

The variables on line 470, TMPR1 and TMPI1 would normally not be used in a BASIC program as more than one operation can be performed on each line. However, indirect table lookups always use a separate line of assembly code, so separate lines have been used here.

Lines 700-810 perform the post-weave. This is not in the flowchart, but can be found in Equation 11. Once again, table look-ups are separated and additional variables are used for clarity. The variables BR(x) are the bit reversal values of x.

Line 830 calculates the magnitude of the harmonic components.

Lines 900-950 print the results of the calculations, with line 900 determining if the print-out should be in hex or decimal.

Lines 1000-1080 are the data for the bit reversal values and input datapoints. The input waveform is one cycle of a square-wave.

7.0 ASM96 PROGRAM FOR FFTS

The BASIC program just presented has been used as an outline for the ASM96 program shown in Listing 2. There are many advantages to using the BASIC program as a model, the main ones being debugging and testing. Since the BASIC program is so similar in program flow to the ASM96 program, it's possible to stop the ASM96 program at almost any point and verify that the results are correct.

MCS-96 MACRO ASSEMBLER FFT_RUN

02/18/86

PAGE 1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:FFTRUN.A96

OBJECT FILE: :F2:FFTRUN.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $pagelength(50)
2
3 FFT_RUN MODULE STACKSIZE(6)
4
5 ; Intel Corporation, January 24, 1986
6 ; by Ira Horden, MCO Applications
7
8
9 ; This module performs a fast fourier transform (FFT) on 64 real data
10 ; points using a 2N-point algorithm. The algorithm involves using a standard
11 ; FFT procedure for 32 real and 32 imaginary numbers. The real and imaginary
12 ; arrays are filled alternately with real data points, and the output of the
13 ; FFT is run through a post-processor. The result is a one sided array with 32
14 ; output buckets. The post processing includes a table lookup algorithm for
15 ; taking the square root of an unsigned 32-bit number.
16
17 ; All of the calculations in the main FFT program are done using 16-bit
18 ; signed integers. The maximum value of any frequency component is therefore
19 ; +/- 32K. (Note that a square wave of +/-32K has a fundamental component
20 ; greater than +/- 40K). Wherever possible tables are used to increase the
21 ; speed of math operations. The complete transform, including obtaining the
22 ; absolute magnitude of each frequency component, executes in 12
23 ; milliseconds with internal variables, 14 ms with external.
24
25 ; The program requires two 32-word input arrays, with the sample values
26 ; alternated between the two. These start at XREAL and XIMAG. The resultant
27 ; magnitude will be placed in a 32-word array at FFT_OUT. These are all
28 ; externally defined variables. The external constant SCALE_FACTOR is used to
29 ; divide the output when averaging will be used. Since the program averages
30 ; its output, it is necessary to clear the array based at FFT_OUT before
31 ; calling FFT_CALC to start the program.
32
33 ; The program was originally written in BASIC for testing purposes. The
34 ; comments include these BASIC statements to make it easier to follow the
35 ; algorithms.
36
37 $EJECT

```

270189-33

MCS-96 MACRO ASSEMBLER FFT_RUN

02/18/86

PAGE 2

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			38	
	0000		39	RSEG
			40	EXTEN port1, zero, error
			41	
	0024		42	OSEG at 24H
	0024		43	TMPI: dsl 1 ; Temporary register, Real
	0028		44	TMPI: dsl 1 ; Temporary register, Imaginary
	002C		45	TMPI: dsl 1 ; Temporary register, Real
	0030		46	TMPI: dsl 1 ; Temporary register, Imaginary
	0034		47	XTMP: dsl 1 ; Temporary data register, Real
	0038		48	XTMP: dsl 1 ; Temporary data register, Imaginary
	003C		49	XRRK: dsl 1
	0040		50	XRRK: dsl 1
	0044		51	XRRK: dsl 1
	0048		52	XRRK: dsl 1
	003C		53	diff equ xrrk :long ; Table difference for square root
	0040		54	sqrt equ xrrnk :long ; Square root
	0040		55	log equ xrrnk :long ; 10 Log magnitude^2
	0044		56	nxtloc equ xirk :long ; Next location in table
			57	
	003C		58	WRP equ xrrk :word ; Multiplication factor, Real
	003E		59	WIP equ xrrk+2 :word ; Multiplication factor, Imaginary
	0040		60	PWR equ xrrnk :word
	0042		61	IN_CNT equ xrrnk+2 :word
	0044		62	NDIV2 equ xirk :word ; n divided by 2 (0 < n < N) #2
			63	
	004C		64	KPTR: dsw 1 ; K for counter #2 to index words
	004E		65	KN2: dsw 1 ; KPTR + NDIV2
	0050		66	N_SUB_K: dsw 1 ; N-K #2 to index words
	0052		67	RK: dsw 1 ; Bit reversed pointer of KPTR
	0054		68	RNK: dsw 1 ; Bit reversed pointer of N_SUB_K
	0056		69	SHIFT_CNT: dsw 1
	0058		70	LOOP_CNT: dsw 1
	004E		71	ptr equ kn2 :word ; Pointer for square root table
	0000		72	DSEG
			73	
			74	EXTRN FFT_MODE ; FFT MODE: mode for FFT input and graphing
			75	EXTRN XREAL, XIMAG ; XREAL, XIMAG: Base addresses for 32 16-bit signed
			76	; entries for real and imaginary numbers respectively.
			77	EXTRN FFT_OUT ; FFT_OUT: Starting address for 32 word array
			78	; of magnitude information.
			79	
	0000		80	OUTR: dsw 32 ; Real component of fft
	0040		81	OUTI: dsw 32 ; Imaginary component of waveform
			82	PUBLIC OUTR,OUTI
			83	
			84	\$EJECT

270189-34

MCS-96 MACRO ASSEMBLER FFT_RUN 02/18/86 PAGE 3

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		85	
		86	CSEG at 2280H
		87	
		88	PUBLIC fft_calc ; Starting point for FFT algorithm
		89	
		90	EXTERN scale_factor ; Shift factor used to prevent overflow when averaging
		91	; fft outputs
		92	
		93	
		94	
		95	FFT_CALC: ; START FOURIER CALCULATIONS
		96	; 400 ' INITIALIZATION OF LOOP
2280	1100	96	clrb error
2282	B10100	97	lcb portl,\$00000001b ;*** Indication Only
		98	
		99	clrvt
2285	FC	100	lcb loop_cnt,\$1
2286	B10158	101	lcb shift_cnt,\$4
2289	B10456	102	ld ndiv2,\$32
228C	A1200044	103	; 410 K=0
		104	OUT_LOOP: ;
2290	950400	105	xorb portl,\$00000100B ;*** Indication Only
2293	014C	106	clr kptr
		107	; 420 IF LOOP > EXP THEN 700
2295	990558	108	cmpb loop_cnt, #5 ; 32=2^5
2298	DA0220A3	109	bgt UNWEAVE
		110	
		111	
229C		112	MID_LOOP: ; 430 INCNT=0
229C	0142	113	clr in_cnt
		114	
		115	; 440 ' CALCULATIONS BEGIN HERE
229E		116	IN_LOOP: ;
229E	65020042	117	add in_cnt,\$2 ; 450 INCNT=INCNT+1
		118	; 460 P=BR(INT(K/(2^SHIFT)))
22A2	A04C40	119	ld pwr,kptr
22A5	085640	120	shr pwr,shift_cnt ; Calculate multiplication factors
22A8	71FE40	121	andb pwr,\$11111110B
22AB	A341003840	122	ld pwr,brev[pwr]
		123	; 470 WRP=WR(P) : WIP=WI(P) : KN2=K+N2
22B0	A34144393C	124	gwr: ld wrp,wr[pwr]
22B5	A34186393E	125	ld wip,wi[pwr]
22BA	44444C4E	126	add kn2,kptr,ndiv2
		127	\$object

270189-35

MCS-96 MACRO ASSEMBLER FFT_RUN

02/18/86

PAGE 4

```

ERR LOC OBJECT          LINE    SOURCE STATEMENT
                                ;; Complex multiplication follows
                                128
                                129
                                130
                                131 ;
                                132 ;
                                133 ;
                                134 ;
                                135 ;
                                136 ;
                                137 ;
                                138 ;
                                139 ;
                                140 ;
                                141 ;
                                142 BVT ERR1 ; Branch on error in complex multiplications
                                143
                                144 ;
                                145 ;
                                146 ;
                                147 ;
                                148 ;
                                149 ;
                                150 ;
                                151 ;
                                152 ;
                                153 ;
                                154 ;
                                155 ;
                                156 ;
                                157 ;
                                158 ;
                                159 ;
                                160 ;
                                161 ;
                                162 BVT ERR2 ; Branch on error in complex additions
                                163
                                164 $eject
                                165
                                166
                                167
                                168
                                169
                                170
                                171
                                172
                                173
                                174
                                175
                                176
                                177
                                178
                                179
                                180
                                181
                                182
                                183
                                184
                                185
                                186
                                187
                                188
                                189
                                190
                                191
                                192
                                193
                                194
                                195
                                196
                                197
                                198
                                199
                                200
                                201
                                202
                                203
                                204
                                205
                                206
                                207
                                208
                                209
                                210
                                211
                                212
                                213
                                214
                                215
                                216
                                217
                                218
                                219
                                220
                                221
                                222
                                223
                                224
                                225
                                226
                                227
                                228
                                229
                                230
                                231
                                232
                                233
                                234
                                235
                                236
                                237
                                238
                                239
                                240
                                241
                                242
                                243
                                244
                                245
                                246
                                247
                                248
                                249
                                250
                                251
                                252
                                253
                                254
                                255
                                256
                                257
                                258
                                259
                                260
                                261
                                262
                                263
                                264
                                265
                                266
                                267
                                268
                                269
                                270
                                271
                                272
                                273
                                274
                                275
                                276
                                277
                                278
                                279
                                280
                                281
                                282
                                283
                                284
                                285
                                286
                                287
                                288
                                289
                                290
                                291
                                292
                                293
                                294
                                295
                                296
                                297
                                298
                                299
                                300
                                301
                                302
                                303
                                304
                                305
                                306
                                307
                                308
                                309
                                310
                                311
                                312
                                313
                                314
                                315
                                316
                                317
                                318
                                319
                                320
                                321
                                322
                                323
                                324
                                325
                                326
                                327
                                328
                                329
                                330
                                331
                                332
                                333
                                334
                                335
                                336
                                337
                                338
                                339
                                340
                                341
                                342
                                343
                                344
                                345
                                346
                                347
                                348
                                349
                                350
                                351
                                352
                                353
                                354
                                355
                                356
                                357
                                358
                                359
                                360
                                361
                                362
                                363
                                364
                                365
                                366
                                367
                                368
                                369
                                370
                                371
                                372
                                373
                                374
                                375
                                376
                                377
                                378
                                379
                                380
                                381
                                382
                                383
                                384
                                385
                                386
                                387
                                388
                                389
                                390
                                391
                                392
                                393
                                394
                                395
                                396
                                397
                                398
                                399
                                400
                                401
                                402
                                403
                                404
                                405
                                406
                                407
                                408
                                409
                                410
                                411
                                412
                                413
                                414
                                415
                                416
                                417
                                418
                                419
                                420
                                421
                                422
                                423
                                424
                                425
                                426
                                427
                                428
                                429
                                430
                                431
                                432
                                433
                                434
                                435
                                436
                                437
                                438
                                439
                                440
                                441
                                442
                                443
                                444
                                445
                                446
                                447
                                448
                                449
                                450
                                451
                                452
                                453
                                454
                                455
                                456
                                457
                                458
                                459
                                460
                                461
                                462
                                463
                                464
                                465
                                466
                                467
                                468
                                469
                                470
                                471
                                472
                                473
                                474
                                475
                                476
                                477
                                478
                                479
                                480
                                481
                                482
                                483
                                484
                                485
                                486
                                487
                                488
                                489
                                490
                                491
                                492
                                493
                                494
                                495
                                496
                                497
                                498
                                499
                                500
                                501
                                502
                                503
                                504
                                505
                                506
                                507
                                508
                                509
                                510
                                511
                                512
                                513
                                514
                                515
                                516
                                517
                                518
                                519
                                520
                                521
                                522
                                523
                                524
                                525
                                526
                                527
                                528
                                529
                                530
                                531
                                532
                                533
                                534
                                535
                                536
                                537
                                538
                                539
                                540
                                541
                                542
                                543
                                544
                                545
                                546
                                547
                                548
                                549
                                550
                                551
                                552
                                553
                                554
                                555
                                556
                                557
                                558
                                559
                                560
                                561
                                562
                                563
                                564
                                565
                                566
                                567
                                568
                                569
                                570
                                571
                                572
                                573
                                574
                                575
                                576
                                577
                                578
                                579
                                580
                                581
                                582
                                583
                                584
                                585
                                586
                                587
                                588
                                589
                                590
                                591
                                592
                                593
                                594
                                595
                                596
                                597
                                598
                                599
                                600
                                601
                                602
                                603
                                604
                                605
                                606
                                607
                                608
                                609
                                610
                                611
                                612
                                613
                                614
                                615
                                616
                                617
                                618
                                619
                                620
                                621
                                622
                                623
                                624
                                625
                                626
                                627
                                628
                                629
                                630
                                631
                                632
                                633
                                634
                                635
                                636
                                637
                                638
                                639
                                640
                                641
                                642
                                643
                                644
                                645
                                646
                                647
                                648
                                649
                                650
                                651
                                652
                                653
                                654
                                655
                                656
                                657
                                658
                                659
                                660
                                661
                                662
                                663
                                664
                                665
                                666
                                667
                                668
                                669
                                670
                                671
                                672
                                673
                                674
                                675
                                676
                                677
                                678
                                679
                                680
                                681
                                682
                                683
                                684
                                685
                                686
                                687
                                688
                                689
                                690
                                691
                                692
                                693
                                694
                                695
                                696
                                697
                                698
                                699
                                700
                                701
                                702
                                703
                                704
                                705
                                706
                                707
                                708
                                709
                                710
                                711
                                712
                                713
                                714
                                715
                                716
                                717
                                718
                                719
                                720
                                721
                                722
                                723
                                724
                                725
                                726
                                727
                                728
                                729
                                730
                                731
                                732
                                733
                                734
                                735
                                736
                                737
                                738
                                739
                                740
                                741
                                742
                                743
                                744
                                745
                                746
                                747
                                748
                                749
                                750
                                751
                                752
                                753
                                754
                                755
                                756
                                757
                                758
                                759
                                760
                                761
                                762
                                763
                                764
                                765
                                766
                                767
                                768
                                769
                                770
                                771
                                772
                                773
                                774
                                775
                                776
                                777
                                778
                                779
                                780
                                781
                                782
                                783
                                784
                                785
                                786
                                787
                                788
                                789
                                790
                                791
                                792
                                793
                                794
                                795
                                796
                                797
                                798
                                799
                                800
                                801
                                802
                                803
                                804
                                805
                                806
                                807
                                808
                                809
                                810
                                811
                                812
                                813
                                814
                                815
                                816
                                817
                                818
                                819
                                820
                                821
                                822
                                823
                                824
                                825
                                826
                                827
                                828
                                829
                                830
                                831
                                832
                                833
                                834
                                835
                                836
                                837
                                838
                                839
                                840
                                841
                                842
                                843
                                844
                                845
                                846
                                847
                                848
                                849
                                850
                                851
                                852
                                853
                                854
                                855
                                856
                                857
                                858
                                859
                                860
                                861
                                862
                                863
                                864
                                865
                                866
                                867
                                868
                                869
                                870
                                871
                                872
                                873
                                874
                                875
                                876
                                877
                                878
                                879
                                880
                                881
                                882
                                883
                                884
                                885
                                886
                                887
                                888
                                889
                                890
                                891
                                892
                                893
                                894
                                895
                                896
                                897
                                898
                                899
                                900
                                901
                                902
                                903
                                904
                                905
                                906
                                907
                                908
                                909
                                910
                                911
                                912
                                913
                                914
                                915
                                916
                                917
                                918
                                919
                                920
                                921
                                922
                                923
                                924
                                925
                                926
                                927
                                928
                                929
                                930
                                931
                                932
                                933
                                934
                                935
                                936
                                937
                                938
                                939
                                940
                                941
                                942
                                943
                                944
                                945
                                946
                                947
                                948
                                949
                                950
                                951
                                952
                                953
                                954
                                955
                                956
                                957
                                958
                                959
                                960
                                961
                                962
                                963
                                964
                                965
                                966
                                967
                                968
                                969
                                970
                                971
                                972
                                973
                                974
                                975
                                976
                                977
                                978
                                979
                                980
                                981
                                982
                                983
                                984
                                985
                                986
                                987
                                988
                                989
                                990
                                991
                                992
                                993
                                994
                                995
                                996
                                997
                                998
                                999
                                1000

```

530-89-90

270189-36

MCS-96 MACRO ASSEMBLER FFT_RUN

02/18/86

PAGE 5

```

ERR LOC OBJECT      LINE      SOURCE STATEMENT
165 ;
166 ik: add kptr,#2      ;;; 560 K=K+1
167
168 ;
169 cmp in_cnt,ndiv2      ;;; 570 IF INCNT<N2 THEN GOTO 450
231F D602277B      170 blt IN_LOOP
171
172 ;
173 add kptr,ndiv2      ;;; 580 K=K+N2
2323 64444C      174 ;
175 cmp kptr,#62      ;;; 590 IF K<N1 THEN GOTO 430
232A D602276E      176 blt MID_LOOP
177
178 ;
179 incb loop_cnt      ;;; 600 LOOP=LOOP+1 : N2=N2/2
232E 1758      180 shra ndiv2,#1      ;;; 605 SHIFT=SHIFT+1
2330 0A0144      181 decb shift_cnt
2333 1556      182 ;
2335 2759      183 br OUT_LOOP      ;;; 610 GOTO 400
2337 B10100      184
233A F0      185
233B B10200      186 ERR1: ldb error,#01 ; overflow error, 1st set of calculations
233E F0      187 ret
233F B10200      188 ERR2: ldb error,#02 ; overflow error, 2nd set of calculations
2340 F0      189 ret
2341 B10200      190
2342 B10200      191 $EJECT
2343 B10200      192
2344 B10200      193
2345 B10200      194
2346 B10200      195
2347 B10200      196
2348 B10200      197
2349 B10200      198
234A B10200      199
234B B10200      200
234C B10200      201
234D B10200      202
234E B10200      203
234F B10200      204
2350 B10200      205
2351 B10200      206
2352 B10200      207
2353 B10200      208
2354 B10200      209
2355 B10200      210
2356 B10200      211
2357 B10200      212
2358 B10200      213
2359 B10200      214
235A B10200      215
235B B10200      216
235C B10200      217
235D B10200      218
235E B10200      219
235F B10200      220
2360 B10200      221
2361 B10200      222
2362 B10200      223
2363 B10200      224
2364 B10200      225
2365 B10200      226
2366 B10200      227
2367 B10200      228
2368 B10200      229
2369 B10200      230
236A B10200      231
236B B10200      232
236C B10200      233
236D B10200      234
236E B10200      235
236F B10200      236
2360 B10200      237
2361 B10200      238
2362 B10200      239
2363 B10200      240
2364 B10200      241
2365 B10200      242
2366 B10200      243
2367 B10200      244
2368 B10200      245
2369 B10200      246
236A B10200      247
236B B10200      248
236C B10200      249
236D B10200      250
236E B10200      251
236F B10200      252
2370 B10200      253
2371 B10200      254
2372 B10200      255
2373 B10200      256
2374 B10200      257
2375 B10200      258
2376 B10200      259
2377 B10200      260
2378 B10200      261
2379 B10200      262
237A B10200      263
237B B10200      264
237C B10200      265
237D B10200      266
237E B10200      267
237F B10200      268
2380 B10200      269
2381 B10200      270
2382 B10200      271
2383 B10200      272
2384 B10200      273
2385 B10200      274
2386 B10200      275
2387 B10200      276
2388 B10200      277
2389 B10200      278
238A B10200      279
238B B10200      280
238C B10200      281
238D B10200      282
238E B10200      283
238F B10200      284
2390 B10200      285
2391 B10200      286
2392 B10200      287
2393 B10200      288
2394 B10200      289
2395 B10200      290
2396 B10200      291
2397 B10200      292
2398 B10200      293
2399 B10200      294
239A B10200      295
239B B10200      296
239C B10200      297
239D B10200      298
239E B10200      299
239F B10200      300
23A0 B10200      301
23A1 B10200      302
23A2 B10200      303
23A3 B10200      304
23A4 B10200      305
23A5 B10200      306
23A6 B10200      307
23A7 B10200      308
23A8 B10200      309
23A9 B10200      310
23AA B10200      311
23AB B10200      312
23AC B10200      313
23AD B10200      314
23AE B10200      315
23AF B10200      316
23B0 B10200      317
23B1 B10200      318
23B2 B10200      319
23B3 B10200      320
23B4 B10200      321
23B5 B10200      322
23B6 B10200      323
23B7 B10200      324
23B8 B10200      325
23B9 B10200      326
23BA B10200      327
23BB B10200      328
23BC B10200      329
23BD B10200      330
23BE B10200      331
23BF B10200      332
23C0 B10200      333
23C1 B10200      334
23C2 B10200      335
23C3 B10200      336
23C4 B10200      337
23C5 B10200      338
23C6 B10200      339
23C7 B10200      340
23C8 B10200      341
23C9 B10200      342
23CA B10200      343
23CB B10200      344
23CC B10200      345
23CD B10200      346
23CE B10200      347
23CF B10200      348
23D0 B10200      349
23D1 B10200      350
23D2 B10200      351
23D3 B10200      352
23D4 B10200      353
23D5 B10200      354
23D6 B10200      355
23D7 B10200      356
23D8 B10200      357
23D9 B10200      358
23DA B10200      359
23DB B10200      360
23DC B10200      361
23DD B10200      362
23DE B10200      363
23DF B10200      364
23E0 B10200      365
23E1 B10200      366
23E2 B10200      367
23E3 B10200      368
23E4 B10200      369
23E5 B10200      370
23E6 B10200      371
23E7 B10200      372
23E8 B10200      373
23E9 B10200      374
23EA B10200      375
23EB B10200      376
23EC B10200      377
23ED B10200      378
23EE B10200      379
23EF B10200      380
23F0 B10200      381
23F1 B10200      382
23F2 B10200      383
23F3 B10200      384
23F4 B10200      385
23F5 B10200      386
23F6 B10200      387
23F7 B10200      388
23F8 B10200      389
23F9 B10200      390
23FA B10200      391
23FB B10200      392
23FC B10200      393
23FD B10200      394
23FE B10200      395
23FF B10200      396
2400 B10200      397
2401 B10200      398
2402 B10200      399
2403 B10200      400
2404 B10200      401
2405 B10200      402
2406 B10200      403
2407 B10200      404
2408 B10200      405
2409 B10200      406
240A B10200      407
240B B10200      408
240C B10200      409
240D B10200      410
240E B10200      411
240F B10200      412
2410 B10200      413
2411 B10200      414
2412 B10200      415
2413 B10200      416
2414 B10200      417
2415 B10200      418
2416 B10200      419
2417 B10200      420
2418 B10200      421
2419 B10200      422
241A B10200      423
241B B10200      424
241C B10200      425
241D B10200      426
241E B10200      427
241F B10200      428
2420 B10200      429
2421 B10200      430
2422 B10200      431
2423 B10200      432
2424 B10200      433
2425 B10200      434
2426 B10200      435
2427 B10200      436
2428 B10200      437
2429 B10200      438
242A B10200      439
242B B10200      440
242C B10200      441
242D B10200      442
242E B10200      443
242F B10200      444
2430 B10200      445
2431 B10200      446
2432 B10200      447
2433 B10200      448
2434 B10200      449
2435 B10200      450
2436 B10200      451
2437 B10200      452
2438 B10200      453
2439 B10200      454
243A B10200      455
243B B10200      456
243C B10200      457
243D B10200      458
243E B10200      459
243F B10200      460
2440 B10200      461
2441 B10200      462
2442 B10200      463
2443 B10200      464
2444 B10200      465
2445 B10200      466
2446 B10200      467
2447 B10200      468
2448 B10200      469
2449 B10200      470
244A B10200      471
244B B10200      472
244C B10200      473
244D B10200      474
244E B10200      475
244F B10200      476
2450 B10200      477
2451 B10200      478
2452 B10200      479
2453 B10200      480
2454 B10200      481
2455 B10200      482
2456 B10200      483
2457 B10200      484
2458 B10200      485
2459 B10200      486
245A B10200      487
245B B10200      488
245C B10200      489
245D B10200      490
245E B10200      491
245F B10200      492
2460 B10200      493
2461 B10200      494
2462 B10200      495
2463 B10200      496
2464 B10200      497
2465 B10200      498
2466 B10200      499
2467 B10200      500
2468 B10200      501
2469 B10200      502
246A B10200      503
246B B10200      504
246C B10200      505
246D B10200      506
246E B10200      507
246F B10200      508
2470 B10200      509
2471 B10200      510
2472 B10200      511
2473 B10200      512
2474 B10200      513
2475 B10200      514
2476 B10200      515
2477 B10200      516
2478 B10200      517
2479 B10200      518
247A B10200      519
247B B10200      520
247C B10200      521
247D B10200      522
247E B10200      523
247F B10200      524
2480 B10200      525
2481 B10200      526
2482 B10200      527
2483 B10200      528
2484 B10200      529
2485 B10200      530
2486 B10200      531
2487 B10200      532
2488 B10200      533
2489 B10200      534
248A B10200      535
248B B10200      536
248C B10200      537
248D B10200      538
248E B10200      539
248F B10200      540
2490 B10200      541
2491 B10200      542
2492 B10200      543
2493 B10200      544
2494 B10200      545
2495 B10200      546
2496 B10200      547
2497 B10200      548
2498 B10200      549
2499 B10200      550
249A B10200      551
249B B10200      552
249C B10200      553
249D B10200      554
249E B10200      555
249F B10200      556
2500 B10200      557
2501 B10200      558
2502 B10200      559
2503 B10200      560
2504 B10200      561
2505 B10200      562
2506 B10200      563
2507 B10200      564
2508 B10200      565
2509 B10200      566
250A B10200      567
250B B10200      568
250C B10200      569
250D B10200      570
250E B10200      571
250F B10200      572
2510 B10200      573
2511 B10200      574
2512 B10200      575
2513 B10200      576
2514 B10200      577
2515 B10200      578
2516 B10200      579
2517 B10200      580
2518 B10200      581
2519 B10200      582
251A B10200      583
251B B10200      584
251C B10200      585
251D B10200      586
251E B10200      587
251F B10200      588
2520 B10200      589
2521 B10200      590
2522 B10200      591
2523 B10200      592
2524 B10200      593
2525 B10200      594
2526 B10200      595
2527 B10200      596
2528 B10200      597
2529 B10200      598
252A B10200      599
252B B10200      600
252C B10200      601
252D B10200      602
252E B10200      603
252F B10200      604
2530 B10200      605
2531 B10200      606
2532 B10200      607
2533 B10200      608
2534 B10200      609
2535 B10200      610
2536 B10200      611
2537 B10200      612
2538 B10200      613
2539 B10200      614
253A B10200      615
253B B10200      616
253C B10200      617
253D B10200      618
253E B10200      619
253F B10200      620
2540 B10200      621
2541 B10200      622
2542 B10200      623
2543 B10200      624
2544 B10200      625
2545 B10200      626
2546 B10200      627
2547 B10200      628
2548 B10200      629
2549 B10200      630
254A B10200      631
254B B10200      632
254C B10200      633
254D B10200      634
254E B10200      635
254F B10200      636
2550 B10200      637
2551 B10200      638
2552 B10200      639
2553 B10200      640
2554 B10200      641
2555 B10200      642
2556 B10200      643
2557 B10200      644
2558 B10200      645
2559 B10200      646
255A B10200      647
255B B10200      648
255C B10200      649
255D B10200      650
255E B10200      651
255F B10200      652
2560 B10200      653
2561 B10200      654
2562 B10200      655
2563 B10200      656
2564 B10200      657
2565 B10200      658
2566 B10200      659
2567 B10200      660
2568 B10200      661
2569 B10200      662
256A B10200      663
256B B10200      664
256C B10200      665
256D B10200      666
256E B10200      667
256F B10200      668
2570 B10200      669
2571 B10200      670
2572 B10200      671
2573 B10200      672
2574 B10200      673
2575 B10200      674
2576 B10200      675
2577 B10200      676
2578 B10200      677
2579 B10200      678
257A B10200      679
257B B10200      680
257C B10200      681
257D B10200      682
257E B10200      683
257F B10200      684
2580 B10200      685
2581 B10200      686
2582 B10200      687
2583 B10200      688
2584 B10200      689
2585 B10200      690
2586 B10200      691
2587 B10200      692
2588 B10200      693
2589 B10200      694
258A B10200      695
258B B10200      696
258C B10200      697
258D B10200      698
258E B10200      699
258F B10200      700
2590 B10200      701
2591 B10200      702
2592 B10200      703
2593 B10200      704
2594 B10200      705
2595 B10200      706
2596 B10200      707
2597 B10200      708
2598 B10200      709
2599 B10200      710
259A B10200      711
259B B10200      712
259C B10200      713
259D B10200      714
259E B10200      715
259F B10200      716
2600 B10200      717
2601 B10200      718
2602 B10200      719
2603 B10200      720
2604 B10200      721
2605 B10200      722
2606 B10200      723
2607 B10200      724
2608 B10200      725
2609 B10200      726
260A B10200      727
260B B10200      728
260C B10200      729
260D B10200      730
260E B10200      731
260F B10200      732
2610 B10200      733
2611 B10200      734
2612 B10200      735
2613 B10200      736
2614 B10200      737
2615 B10200      738
2616 B10200      739
2617 B10200      740
2618 B10200      741
2619 B10200      742
261A B10200      743
261B B10200      744
261C B10200      745
261D B10200      746
261E B10200      747
261F B10200      748
2620 B10200      749
2621 B10200      750
2622 B10200      751
2623 B10200      752
2624 B10200      753
2625 B10200      754
2626 B10200      755
2627 B10200      756
2628 B10200      757
2629 B10200      758
262A B10200      759
262B B10200      760
262C B10200      761
262D B10200      762
262E B10200      763
262F B10200      764
2630 B10200      765
2631 B10200      766
2632 B10200      767
2633 B10200      768
2634 B10200      769
2635 B10200      770
2636 B10200      771
2637 B10200      772
2638 B10200      773
2639 B10200      774
263A B10200      775
263B B10200      776
263C B10200      777
263D B10200      778
263E B10200      779
263F B10200      780
2640 B10200      781
2641 B10200      782
2642 B10200      783
2643 B10200      784
2644 B10200      785
2645 B10200      786
2646 B10200      787
2647 B10200      788
2648 B10200      789
2649 B10200      790
264A B10200      791
264B B10200      792
264C B10200      793
264D B10200      794
264E B10200      795
264F B10200      796
2650 B10200      797
2651 B10200      798
2652 B10200      799
2653 B10200      800
2654 B10200      801
2655 B10200      802
2656 B10200      803
2657 B10200      804
2658 B10200      805
2659 B10200      806
265A B10200      807
265B B10200      808
265C B10200      809
265D B10200      810
265E B10200      811
265F B10200      812
2660 B10200      813
2661 B10200      814
2662 B10200      815
2663 B10200      816
2664 B10200      817
2665 B10200      818
2666 B10200      819
2667 B10200      820
2668 B10200      821
2669 B10200      822
266A B10200      823
266B B10200      824
266C B10200      825
266D B10200      826
266E B10200      827
266F B10200      828
2670 B10200      829
2671 B10200      830
2672 B10200      831
2673 B10200      832
2674 B10200      833
2675 B10200      834
2676 B10200      835
2677 B10200      836
2678 B10200      837
2679 B10200      838
267A B10200      839
267B B10200      840
267C B10200      841
267D B10200      842
267E B10200      843
267F B10200      844
2680 B10200      845
2681 B10200      846
2682 B10200      847
2683 B10200      848
2684 B10200      849
2685 B10200      850
2686 B10200      851
2687 B10200      852
2688 B10200      853
2689 B10200      854
268A B10200      855
268B B10200      856
268C B10200      857
268D B10200      858
268E B10200      859
268F B10200      860
2690 B10200      861
2691 B10200      862
2692 B10200      863
2693 B10200      864
2694 B10200      865
2695 B10200      866
2696 B10200      867
2697 B10200      868
2698 B10200      869
2699 B10200      870
269A B10200      871
269B B10200      872
269C B10200      873
269D B10200      874
269E B10200      875
269F B10200      876
2700 B10200      877
2701 B10200      878
2702 B10200      879
2703 B10200      880
2704 B10200      881
2705 B10200      882
2706 B10200      883
2707 B10200      884
2708 B10200      885
2709 B10200      886
270A B10200      887
270B B10200      888
270C B10200      889
270D B10200      890
270E B10200      891
270F B10200      892
2710 B10200      893
2711 B10200      894
2712 B10200      895
2713 B10200      896
2714 B10200      897
2715 B10200      898
2716 B10200      899
2717 B10200      900
2718 B10200      901
2719 B10200      902
271A B10200      903
271B B10200      904
271C B10200      905
271D B10200      906
271E B10200      907
271F B10200      908
2720 B10200      909
2721 B10200      910
2722 B10200      911
2723 B10200      912
2724 B10200      913
2725 B10200      914
2726 B10200      915
2727 B10200      916
2728 B10200      917
2729 B10200      918
272A B10200      919
272B B10200      920
272C B10
```

MCS-96 MACRO ASSEMBLER FFT_RUN 02/18/86 PAGE 6

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			192	
			193	; ; ; 700 ' POST-PROCESSING AND REORDERING STARTS HERE
	233F		194	UNWEAVE:
	233F	B10200	195	ld port1,#00000010b ;****
			196	
			197	; ; ; 720 FOR K=0 TO 31
	2342	014C	198	clr kptr
	2344	A1400050	199	ld n_sub_k,#64
	2348		200	UN_LOOP:
			201	; ; ; 740 XIBRK=XI(BR(K)) : XBRNK=XR(BR(K))
	2348	A34D003852	202	ld rk,brev[kptr]
	234D	A35300003C	203	ld xrrk,xreal[rk]
	2352	063C	204	ext xrrk
	2354	A353000044	205	ld xirk,ximag[rk]
	2359	0644	206	ext xirk
			207	; ; ; 750 XIBRNK=XI(BR(N-K)):XBRNK=XR(BR(N-K))
	2358	A351003854	208	ld rnk,brev[n_sub_k]
	2360	A355000040	209	ld xrrnk,xreal[rnk]
	2365	0640	210	ext xrrnk
	2367	A355000048	211	ld xirnk,ximag[rnk]
	236C	0648	212	ext xirnk
			213	; ; ; 760 TI=(XIBRK + XBRNK)/2
	236E	44484428	214	ar: add tmpi,xirk,xirnk
	2372	A04A2A	215	ld tmpi+2,xirnk+2
	2375	A4462A	216	addc tmpi+2,xirnk+2
	2378	0E0128	217	shrl tmpi,#1 ; 16 bit result in tmpi
			218	
			219	; ; ; 770 TR=(XIBRK - XBRNK)/2
	237B	48403C24	220	sub tmpr,xrrk,xrrnk
	237F	A03E26	221	ld tmpr+2,xrrk+2
	2382	A84226	222	subc tmpr+2,xrrnk+2
	2385	0E0124	223	shrl tmpr,#1 ; 16 bit result in tmpr
			224	
			225	; ; ; 780 XRT= (XIBRK + XBRNK)/4
	2388	44403C34	226	add xrtmp,xrrk,xrrnk
	238C	A03E36	227	ld xrtmp+2,xrrk+2
	238F	A44236	228	addc xrtmp+2,xrrnk+2
	2392	0D0E34	229	shll xrtmp,#14 ; 32 bit result in xrtmp
			230	
			231	; ; ; 790 XIT= (XIBRK-XIBRNK)/4
	2395	48484438	232	sub xitmp,xirk,xirnk
	2399	A0463A	233	ld xitmp+2,xirk+2
	239C	A84A3A	234	subc xitmp+2,xirnk+2
	239F	0D0E38	235	shll xitmp,#14 ; 32 bit result in xitmp
			236	
			237	\$eject

270189-38

MCS-96 MACRO ASSEMBLER FFT_RUN 02/18/86 PAGE 7

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		238	;
		239	;
		240	;
		241	800 OUTR= (XRT + TI*COSFN(K)/2 - TR*SINFN(K)/2)
		242	mr: mul tmp1,tmp1,sinfn[kptr]
		243	mul tmp1,tmp1,cosfn[kptr]
		244	add xrtmp,tmp1
		245	addc xrtmp+2,tmp1+2
		246	sub xrtmp,tmp1
		247	subc xrtmp+2,tmp1+2
		248	st xrtmp+2,outr[kptr] ;; OUTR = Real Output Values
		249	
		250	
		251	;
		252	810 OUTI= (XIT - TI*SINFN(K)/2 - TR*COSFN(K)/2)
		253	mi: mul tmp1,tmp1,cosfn[kptr]
		254	mul tmp1,tmp1,sinfn[kptr]
		255	sub xitmp,tmp1
		256	subc xitmp+2,tmp1+2
		257	sub xitmp,tmp1
		258	subc xitmp+2,tmp1+2
		259	st xitmp+2,outi[kptr] ;; OUTI = Imaginary Output values
		260	
		261	
		262	830 MAG =SQR(OUTR*OUTR + OUTI*OUTI)
		263	;
		264	GET_MAG: ;; Get Magnitude of Vector
		265	ld tmp1,xrtmp+2
		266	ld tmp1,xitmp+2
		267	
		268	mul tmp1,tmp1 ; tmp1 = tmp1**2 + tmp1**2
		269	mul tmp1,tmp1
		270	add tmp1,tmp1
		271	addc tmp1+2,tmp1+2
		272	
		273	bcc FFT_MODE,2,CALC_SQRT
		274	
		275	\$eject

270189-39

MCS-96 MACRO ASSEMBLER FFT_RUN

02/18/86

PAGE 8

```

ERR LOC OBJECT          LINE    SOURCE STATEMENT
                                276
                                277      ;;; *** CALCULATE 10 log magnitude^2 ***
                                278      ; Output = 512*10*LOG(x)  x=1,2,3 ... 64K
                                279
                                280      CALC_LOG:
                                281          clr      shft_cnt
                                282          normal   tmpr,shft_cnt  ; Normalize and get normalization factor
                                283          cmpb     shft_cnt,$15
                                284          jle      LOG_IN_RANGE  ; Jump if SHIFT_CNT <= 15
                                285
                                286          clr      log
                                287          br       LOG_STORE
                                288
                                289      LOG_IN_RANGE:
                                290          add      shft_cnt,shft_cnt,shft_cnt  ; Make shift_cnt a pointer
                                291
                                292          ldbz     ptr,tmp+3  ; Most significant byte is table pointer
                                293          add      ptr,ptr,ptr
                                294          add      ptr,$ LOG_TABLE-256  ; ptr= Table + offset (offset=tmp+3)
                                295          ; Use -256 since tmp+3 is always >= 128
                                296          ld       log,[ptr]+
                                297          ld       nxtloc,[ptr]  ; Linear Interpolation
                                298
                                299          sub      nxtloc,log  ; nxtloc = next log - log
                                300
                                301          ldbz     diff,tmp+2  ; diff+1 = nxtloc * tmp+2 / 256
                                302          mulu     diff,nxtloc
                                303
                                304          shr      diff,$8  ; log = log + diff/256
                                305          add      log,diff
                                306          shr      log,$5  ; 8192/32 * 20LOG(x) = 256 * 20LOG(x)
                                307
                                308          addc     log,log_offset[shft_cnt]  ; add log of normalization factor
                                309
                                310          ; Log (M*N) = Log M + Log N
                                311
                                312      LOG_STORE:
                                313          shr      log,$SCALE_FACTOR
                                314          addc     log,zero  ; Divide to prevent overflow during
                                315          add      log,FFT_OUT[kptr]  ; averaging of outputs
                                316          st       log,FFT_OUT[kptr]
                                317
                                318          BR       ENDL
                                319      $reject

```

270189-40

MCS-96 MACRO ASSEMBLER FFT_RUN

02/18/86

PAGE 9

```

ERR LOC OBJECT          LINE    SOURCE STATEMENT
                                320
                                321 CALC_SQRT:          ;;;; *** CALCULATE SQUARE ROOT ***
                                322
2443                                323 clr      shift_cnt
2445 0156                   324 norml   tmpr,shift_cnt ; Normalize and get normalization factor
2445 0F5624                325
2448 D705                   326 jne      SQRT_IN_RANGE ; Jump if tmpr > 0
244A C04200                327 st       zero,sqrt+2
244D 2029                   328 br       SQRT_STORE
                                329
244F                                330 SQRT_IN_RANGE:
244F AC274E                331 ldbze   ptr,tmpr+3 ; Most significant byte is table pointer
2452 444E4E4E             332 add     ptr,ptr,ptr
2456 6508394E             333 add     ptr,#SQ_TABLE-256 ; ptr= Table + offset (offset=tmpr+3)
                                334 ; Use -256 since tmpr+3 is always >= 128
245A A24F40               335 ld       sqrt,[ptr]+
245D A24E44               336 ld       nextloc,[ptr] ; Linear Interpolation
                                337
2460 684044               338 sub     nextloc,sqrt ; nextloc = sqrt - next sqrt
                                339
2463 AC263C               340 ldbze   diff,tmpr+2 ; diff+1 = nextloc * tmpr+2 / 256
2466 6C443C               341 mulu    diff,nextloc
                                342
2469 AC3D3C               343 ldbze   diff,diff+1 ; sqrt = sqrt + delta (diff < 0FFFH)
246C 643C40               344 add     sqrt,diff
                                345
246F 44565656             346 add     shift_cnt,shift_cnt,shift_cnt
                                347
2473 6F57C83940           348 mulu    sqrt,tab_sqr[shift_cnt] ; divide by normalization factor
                                349
                                350 ; mulu acts as divide since if tab2=0FFFFH
                                351 ; sqrt would remain essentially unchanged
2478                                352 SQRT_STORE:
2478 080042                353 shr     sqrt+2,SCALE_FACTOR
247B A40042                354 addc    sqrt+2,zero ; Divide to prevent overflow during
247E 674D000042           355 add     sqrt+2,FFT_OUT[kptr] ; averaging of outputs
2483 C34D000042           356 st      sqrt+2,FFT_OUT[kptr]
                                357
2483                                358 ;;;; *** END OF LOOP ***
                                359
2488 6502004C             360 ;;;; 950 NEXT K
248C 69020050             361 ENDL:   add     kptr,#2
2490 DF0226B4             362 sub     n_sub_k,#2
                                363 bne     UN_LOOP
                                364
2494 F0                   365 RET
                                366 $eject

```

270189-41

MCS-96 MACRO ASSEMBLER FFT_RUN

02/18/86

PAGE 10

```

ERR LOC OBJECT          LINE SOURCE STATEMENT
                                367 ;$nolist
                                368 CSEG AT 3800H      ;;; Use 2k for tables
                                369
                                370 BREV:      ; 2*bit reversal value
                                371
                                372 DCW      2*0, 2*16, 2*8, 2*24, 2*4, 2*20, 2*12, 2*28
                                373 DCW      2*2, 2*18, 2*10, 2*26, 2*6, 2*22, 2*14, 2*30
                                374 DCW      2*1, 2*17, 2*9, 2*25, 2*5, 2*21, 2*13, 2*29
                                375 DCW      2*3, 2*19, 2*11, 2*27, 2*7, 2*23, 2*15, 2*31
                                376
                                377 SINFN:
                                378 DCW      0, 3212, 6393, 9512, 12539, 15446, 18204, 20787
                                379 DCW      23170, 25329, 27245, 28898, 30273, 31356, 32137, 32609
                                380 DCW      32767, 32609, 32137, 31356, 30273, 28898, 27245, 25329
                                381 DCW      23170, 20787, 18204, 15446, 12539, 9512, 6393, 3212
                                382 DCW      0, -3212, -6393, -9512, -12539, -15446, -18204, -20787
                                383 DCW      -23170, -25329, -27245, -28898, -30273, -31356, -32137, -32609
                                384 DCW      -32767, -32609, -32137, -31356, -30273, -28898, -27245, -25329
                                385 DCW      -23170, -20787, -18204, -15446, -12539, -9512, -6393, -3212
                                386 DCW      0
                                387
                                388 COSFN:
                                389 DCW      32767, 32609, 32137, 31356, 30273, 28898, 27245, 25329
                                390 DCW      23170, 20787, 18204, 15446, 12539, 9512, 6393, 3212
                                391 DCW      0, -3212, -6393, -9512, -12539, -15446, -18204, -20787
                                392 DCW      -23170, -25329, -27245, -28898, -30273, -31356, -32137, -32609
                                393 DCW      -32767, -32609, -32137, -31356, -30273, -28898, -27245, -25329
                                394 DCW      -23170, -20787, -18204, -15446, -12539, -9512, -6393, -3212
                                395 DCW      0, 3212, 6393, 9512, 12539, 15446, 18204, 20787
                                396 DCW      23170, 25329, 27245, 28898, 30273, 31356, 32137, 32609
                                397 DCW      32767
                                398
                                399 WR:      ;;; WR = COS(K*2PI/N)
                                400 DCW      32767, 32137, 30273, 27245, 23170, 18204, 12539, 6393
                                401 DCW      0, -6393, -12539, -18204, -23170, -27245, -30273, -32137
                                402 DCW      -32767, -32137, -30273, -27245, -23170, -18204, -12539, -6393
                                403 DCW      0, 6393, 12539, 18204, 23170, 27245, 30273, 32137
                                404 DCW      32767
                                405
                                406 WI:      ;;; WI = -SIN(K*2PI/N)
                                407 DCW      -0, -6393, -12539, -18204, -23170, -27245, -30273, -32137
                                408 DCW      -32767, -32137, -30273, -27245, -23170, -18204, -12539, -6393
                                409 DCW      0, 6393, 12539, 18204, 23170, 27245, 30273, 32137
                                410 DCW      32767, 32137, 30273, 27245, 23170, 18204, 12539, 6393
                                411 DCW      0
                                412 $eject

```

270189-42

MCS-96 MACRO ASSEMBLER FFT_RUN

02/18/86

PAGE 11

```

ERR LOC OBJECT LINE SOURCE STATEMENT
413
414
39C8 415 TAB_SQR: ; 65535/(square root of 2**SHFT_CNT) ; 0<=SHFT_CNT<32
416
417 ;; 1 2 4 8 16 32 64 128
39C8 FFFF04B50080825A 418 DCW 65535, 46340, 32768, 23170, 16384, 11585, 8192, 5793
419
420 ;; 256 512 1024 2048 4096 8192 16384 32768
39D8 001050B0008A805 421 DCW 4096, 2896, 2048, 1448, 1024, 724, 512, 362
422
423 ;; 65536, 131072, 262144, 524288, ...
39E8 0001B5008005B00 424 DCW 256, 181, 128, 91, 64, 45, 32, 23
39F8 10000B000800600 425 DCW 16, 11, 8, 6, 4, 3, 2, 1
426
427
428 SQ_TABLE: ; square root of n * 2**24 N=128, 129, 130 ... 255
429
430 46341, 46522, 46702, 46881, 47059, 47237, 47415, 47591
3A08 05B5BAB56EB621B7 430 DCW
3A18 97BA46B8F5BBA3BC 431 DCW 47767, 47942, 48117, 48291, 48465, 48637, 48809, 48981
3A28 00C0AAC054C1FDC1 432 DCW 49152, 49322, 49492, 49661, 49830, 49998, 50166, 50332
3A38 43C5E9C58BC633C7 433 DCW 50499, 50665, 50830, 50995, 51159, 51323, 51486, 51649
3A48 63CA04CBACB46CC 434 DCW 51811, 51972, 52134, 52294, 52454, 52614, 52773, 52932
3A58 62CF00D09DD03AD1 435 DCW 53090, 53248, 53405, 53562, 53719, 53874, 54030, 54185
3A68 44D4BED477B511D6 436 DCW 54340, 54494, 54647, 54801, 54954, 55106, 55258, 55410
3A78 09D9A0D936DACDA 437 DCW 55561, 55712, 55862, 56012, 56162, 56311, 56459, 56608
3A88 B4DD47DED8DEGDF 438 DCW 56756, 56903, 57051, 57198, 57344, 57490, 57636, 57781
3A98 46E2D7E267E3F7E3 439 DCW 57926, 58071, 58215, 58359, 58503, 58646, 58789, 58931
3AA8 C1B64FE7DD76AE8 440 DCW 59073, 59215, 59357, 59498, 59639, 59779, 59919, 60059
3AB8 27EBB2EB3DECC7EC 441 DCW 60199, 60338, 60477, 60615, 60754, 60891, 61029, 61166
3AC8 77EF00F088F010F1 442 DCW 61303, 61440, 61576, 61712, 61848, 61984, 62119, 62254
3AD8 B4F33BF4C1F446F5 443 DCW 62388, 62523, 62657, 62790, 62924, 63057, 63190, 63323
3AE8 DFF763F87F86AF9 444 DCW 63455, 63587, 63719, 63850, 63982, 64113, 64243, 64374
3AF8 F8FB7AFCFBFC7D7D 445 DCW 64504, 64634, 64763, 64893, 65022, 65151, 65280, 65408
446
447 $reject
3B08 0000000000000000 447 DCW
3B18 0000000000000000 448 DCW
3B28 0000000000000000 449 DCW
3B38 0000000000000000 450 DCW
3B48 0000000000000000 451 DCW
3B58 0000000000000000 452 DCW
3B68 0000000000000000 453 DCW
3B78 0000000000000000 454 DCW
3B88 0000000000000000 455 DCW
3B98 0000000000000000 456 DCW
3BA8 0000000000000000 457 DCW
3BB8 0000000000000000 458 DCW
3BC8 0000000000000000 459 DCW
3BD8 0000000000000000 460 DCW
3BE8 0000000000000000 461 DCW
3BF8 0000000000000000 462 DCW
3C08 0000000000000000 463 DCW
3C18 0000000000000000 464 DCW
3C28 0000000000000000 465 DCW
3C38 0000000000000000 466 DCW
3C48 0000000000000000 467 DCW
3C58 0000000000000000 468 DCW
3C68 0000000000000000 469 DCW
3C78 0000000000000000 470 DCW
3C88 0000000000000000 471 DCW
3C98 0000000000000000 472 DCW
3CA8 0000000000000000 473 DCW
3CB8 0000000000000000 474 DCW
3CC8 0000000000000000 475 DCW
3CD8 0000000000000000 476 DCW
3CE8 0000000000000000 477 DCW
3CF8 0000000000000000 478 DCW
3D08 0000000000000000 479 DCW
3D18 0000000000000000 480 DCW
3D28 0000000000000000 481 DCW
3D38 0000000000000000 482 DCW
3D48 0000000000000000 483 DCW
3D58 0000000000000000 484 DCW
3D68 0000000000000000 485 DCW
3D78 0000000000000000 486 DCW
3D88 0000000000000000 487 DCW
3D98 0000000000000000 488 DCW
3DA8 0000000000000000 489 DCW
3DB8 0000000000000000 490 DCW
3DC8 0000000000000000 491 DCW
3DD8 0000000000000000 492 DCW
3DE8 0000000000000000 493 DCW
3DF8 0000000000000000 494 DCW
3E08 0000000000000000 495 DCW
3E18 0000000000000000 496 DCW
3E28 0000000000000000 497 DCW
3E38 0000000000000000 498 DCW
3E48 0000000000000000 499 DCW
3E58 0000000000000000 500 DCW
3E68 0000000000000000 501 DCW
3E78 0000000000000000 502 DCW
3E88 0000000000000000 503 DCW
3E98 0000000000000000 504 DCW
3EA8 0000000000000000 505 DCW
3EB8 0000000000000000 506 DCW
3EC8 0000000000000000 507 DCW
3ED8 0000000000000000 508 DCW
3EE8 0000000000000000 509 DCW
3EF8 0000000000000000 510 DCW
3F08 0000000000000000 511 DCW
3F18 0000000000000000 512 DCW
3F28 0000000000000000 513 DCW
3F38 0000000000000000 514 DCW
3F48 0000000000000000 515 DCW
3F58 0000000000000000 516 DCW
3F68 0000000000000000 517 DCW
3F78 0000000000000000 518 DCW
3F88 0000000000000000 519 DCW
3F98 0000000000000000 520 DCW
3FA8 0000000000000000 521 DCW
3FB8 0000000000000000 522 DCW
3FC8 0000000000000000 523 DCW
3FD8 0000000000000000 524 DCW
3FE8 0000000000000000 525 DCW
3FF8 0000000000000000 526 DCW

```

270189-43

MCS-96 MACRO ASSEMBLER FFT_RUN

02/18/86

PAGE 12

```

ERR LOC OBJECT LINE SOURCE STATEMENT
3B08 448
449 LOG_TABLE: ; 16384*10*LOG(n/128) n=128,129,130 ... 256
450
3B08 00002A024F047006 451 DCW 0, 554, 1103, 1648, 2190, 2727, 3260, 3789
3B18 DA10E312E9148A16 452 DCW 4314, 4835, 5353, 5866, 6376, 6863, 7386, 7886
3B28 BD20A92292247826 453 DCW 8381, 8873, 9362, 9848, 10330, 10810, 11286, 11758
3B38 C42F973166333335 454 DCW 12228, 12695, 13168, 13619, 14076, 14531, 14983, 15432
3B48 063EC13F7A413043 455 DCW 15878, 16321, 16762, 17200, 17635, 18067, 18497, 18925
3B58 954B3C4DDF4E8150 456 DCW 19349, 19772, 20191, 20609, 21024, 21436, 21846, 22254
3B68 8458175AA85B365D 457 DCW 22660, 23063, 23464, 23862, 24259, 24653, 25045, 25435
3B78 DE646066E0675D69 458 DCW 25822, 26208, 26592, 26973, 27353, 27730, 28106, 28479
3B88 E370247294730275 459 DCW 28851, 29220, 29588, 29954, 30318, 30680, 31040, 31399
3B98 0B7C8E7DCF7E2F80 460 DCW 31755, 32110, 32463, 32815, 33166, 33512, 33859, 34203
3BA8 F28647889B89ED8A 461 DCW 34546, 34887, 35227, 35565, 35902, 36236, 36570, 36901
3BB8 7091B8927F934595 462 DCW 37232, 37560, 37887, 38213, 38537, 38860, 39181, 39501
3BC8 8B9BC89C049E3E9F 463 DCW 39819, 40136, 40452, 40766, 41079, 41390, 41700, 42009
3BD8 4C457E46A7DEA8 464 DCW 42316, 42622, 42927, 43230, 43533, 43833, 44133, 44431
3BE8 B9A8E0AF07B12CB2 465 DCW 44729, 45024, 45319, 45612, 45905, 46196, 46486, 46774
3BF8 D6B7F48811BA2DB8 466 DCW 47062, 47348, 47633, 47917, 48200, 48482, 48763, 49042
3C08 ASC0 467 DCW 49321
468
3C0A LOG_OFFSET: ; 512*10*LOG(2*(15-n)) n= 0,1,2,3 ... 15
469 ; 512*10*LOG(0.5) n= 16,17,18 ... 31
470
471
3C0A 4F5A4A54454E3F48 472 DCW 23119, 21578, 20037, 18495, 16954, 15413, 13871, 12330
3C1A 252A20241A1E1518 473 DCW 10789, 9248, 7706, 6165, 4624, 3083, 1541, 0
474
3C2A 475 END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

270189-44

The BASIC program is used as comments in the ASM96 program. Some of the variables in the ASM96 program have slightly different names than their counterparts in the BASIC program. This was to make the comments fit into the ASM96 code. Highlights in this section of code are a table driven square root routine and log conversion routine which can easily be adapted for use by any program.

Both the square root routine and the log conversion routine use the 32-bit value in the variable TMPR. The square root routine calculates the square root of that value in the variable SQRT+2, a 16-bit variable. In this program, the square root value is averaged and stored in a table.

The log conversion routine divides the value in TMPR by 65536 (2^{16}) and uses table lookup to provide the common log. The result is a 16-bit number with the value $512 * 10 \text{ Log}(\text{TMPR}/65536)$ stored in the variable LOG. This calculation is used to present the results of the FFT in decibels instead of magnitude. With an input of 63095, the output is $512 * 48 \text{ dB}$. The graph program, (Section 10), prints the output value of the plot as $\text{INPUT}/512 \text{ dB}$.

The following descriptions of the ASM code point out some of the highlights and not-so-obvious coding:

Lines 1-104 initialize the code and declare variables. The input and output arrays of the program are declared external. Note that many of the registers are

overlayable, use caution when implementing this routine with others with overlayable registers.

Lines 116-124 calculate the power of W to be used. Note that KPTR is always incremented by 2. The multiple right shift followed by the AND mask creates an even address and the indirect look to the BR (Bit Reversal) table quickly calculates the power PWR.

Lines 130-138 perform the complex multiplications. Since WIP and WRP range from -32767 to $+32767$, the multiplication is easy to handle. The automatic divide by two which occurs when using the upper word only of the 32-bit result is a feature in this case.

Lines 144-163 use right shifts for a fast divide, then add or subtract the desired variables and store them in the array. Note that the upper word of TMPR and TMPI is used, and the same array is used for both the input and output of the operations.

Lines 165-189 update the loop variables and then check for errors on the complex multiplications and additions. If there are no overflows at this time the data will run smoothly through the rest of the program.

Lines 200-212 load variables with values based on the bit reversed values of pointers.

Lines 214-236 perform additions and subtractions to prepare for the next set of formulas. Note that XITMP and XRTMP are 32-bit values.

Lines 240-260 perform multiplies and summations resulting in 32-bit variables. This saves a bit or two of accuracy. The upper words are then stored as the results.

Lines 263-272 generate the squared magnitude of the harmonic component as a 32-bit value.

Lines 278-310 calculate 10 Log (TMPR/65536). The 32-bit register TMPR is divided by 65536 so that the output range would be reasonable.

First, the number is normalized. (It is shifted left until a 1 is in the most significant bit, the number of shifts required is placed in SHFT_CNT.) If it had to be shifted more than 15 times the output is set to zero.

Next, the most significant BYTE is used as a reference for the look-up table, providing a 16-bit result. The next most significant BYTE is then used to perform linear interpolation between the referenced table value and the one above it. The interpolated value is added to the directly referenced one.

The 16-bit result of this table look-up and interpolation is then added to the Log of the normalization factor, which is also stored in a table. This table look-up approach works fast and only uses 290 bytes of table space.

Lines 321-357 calculate the square root of the 32-bit register TMPR using a table look-up approach.

First, the number is normalized. Next, the most significant BYTE is used as a reference for the look-up table, providing a 16-bit result. The next most significant BYTE is then used to perform linear interpolation between the referenced table value and the one above it. The interpolated value is added to the directly referenced one.

The 16-bit result of this table look-up and interpolation is then divided by the square root of the normalization factor, which is also stored in a table. This table look-up approach works fast and only uses 320 bytes of table space. The results are valid to near 14-bits, more than enough for the FFT algorithm.

Lines 352-360 average the magnitude value, if multiple passes are being performed, and then store the value in the array. The loop-counters are incremented and the process repeats itself.

This concludes the FFT routine. In order to use it, it must be called from a main program. The details for calling this routine are covered in the next section.

8.0 BACKGROUND CONTROL PROGRAM

The main routine is shown in Listing 3. It begins with declarations that can be used in almost any program. Note that these are similar, but not identical, to other 8096 include files that have been published. Comments on controlling the Analog to Digital converter routine follow the declarations.

MCS-96 MACRO ASSEMBLER FFT_MAIN.APNOTE

02/18/86

PAGE 1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:FTMAIN.A96

OBJECT FILE: :F2:FTMAIN.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```

ERR LOC OBJECT          LINE    SOURCE STATEMENT
0000          1          $pagelength(50)
0000          2
0000          3          FFT_MAIN.APNOTE MODULE MAIN, STACKSIZE(6)
0000          4
0000          5          ; Intel Corporation, January 24, 1986
0000          6          ; by Ira Horden, MCO Applications
0000          7
0000          8
0000          9          ; This program performs an FFT on real data and plots it on a printer.
0000         10          ; It uses the program modules A2DCON, PLOTSP, and FFTRUN. The adjustable
0000         11          ; parameters of each of the programs are set by this main module.
0000         12
0000         13
0000         14          $INCLUDE (:F0:DEMO96.INC)          ; Include SFR definitions
0000         15          ;$nolist ; Turn listing off for include file
0000         16
0000         17          ;*****
0000         18          ;
0000         19          ; Copyright 1985, Intel Corporation
0000         20          ; October 28, 1985
0000         21          ; by Ira Horden, MCO Applications
0000         22          ;
0000         23          ; DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096
0000         24          ;
0000         25          ;*****
0000         26          ;
0000         27          ZERO          EQU    00h:WORD      ; R/W Zero Register
0000         28          AD_COMMAND    EQU    02h:BYTE      ; W A to D command register
0000         29          AD_RESULT_LO  EQU    02h:BYTE      ; R Low byte of result and channel
0000         30          AD_RESULT_HI  EQU    03h:BYTE      ; R High byte of result
0000         31          HSI_MODE     EQU    03h:BYTE      ; W Controls HSI transition detector
0000         32          HSO_TIME     EQU    04h:WORD      ; W HSI time tag
0000         33          HSI_TIME     EQU    04h:WORD      ; R HSO time tag
0000         34          HSO_COMMAND  EQU    06h:BYTE      ; W HSO command tag
0000         35          HSI_STATUS   EQU    06h:BYTE      ; R HSI status register (reads fifo)
0000         36          SBUF         EQU    07h:BYTE      ; R/W Serial port buffer
0000         37          INT_MASK     EQU    08h:BYTE      ; R/W Interrupt mask register
0000         38          INT_PENDING  EQU    09h:BYTE      ; R/W Interrupt pending register
0000         39          SPCON        EQU    11h:BYTE      ; W Serial port control register
0000         40          SPSTAT       EQU    11h:BYTE      ; R Serial port status register
0000         41          WATCHDOG    EQU    0Ah:BYTE      ; W Watchdog timer

```

270189-45

MCS-96 MACRO ASSEMBLER FFT_MAIN_APNOTE

02/18/86

PAGE 2

```

ERR LOC OBJECT      LINE      SOURCE STATEMENT
000A                =1 42  TIMER1      EQU 0AH:WORD      ; R   Timer1 register
000C                =1 43  TIMER2      EQU 0CH:WORD      ; R   Timer2 register
000E                =1 44  PORT0       EQU 0EH:BYTE      ; R   I/O port 0
000E                =1 45  BAUD_REG    EQU 0EH:BYTE      ; W   Baud rate register
000F                =1 46  PORT1      EQU 0FH:BYTE      ; R/W  I/O port 1
0010                =1 47  PORT2      EQU 10H:BYTE      ; R/W  I/O port 2
0015                =1 48  IOC0       EQU 15H:BYTE      ; W   I/O control register 0
0015                =1 49  IOS0       EQU 15H:BYTE      ; R   I/O status register 0
0016                =1 50  IOC1       EQU 16H:BYTE      ; W   I/O control register 1
0016                =1 51  IOS1       EQU 16H:BYTE      ; R   I/O status register 1
0017                =1 52  PWM_CONTROL EQU 17H:BYTE      ; W   PWM control register
0018                =1 53  SP         EQU 18H:WORD      ; R/W  System stack pointer
000D                =1 54
000D                =1 55  CR         EQU 0DH          ;
000A                =1 56  LF         EQU 0AH          ;
000A                =1 57
000A                =1 58  PUBLIC ZERO,AD_COMMAND,AD_RESULT_LO,AD_RESULT_HI,HSI_MODE,HSO_TIME,HSI_TIME
000A                =1 59  PUBLIC HSO_COMMAND
000A                =1 60  PUBLIC HSI_STATUS,SBUF,INT_MASK,INT_PENDING,WATCHDOG,TIMER1,TIMER2
000A                =1 61  PUBLIC BAUD_REG, PORT0, PORT1, PORT2,SPSTAT,SPCON,IOC0,IOC1,IOS0,IOS1
000A                =1 62  PUBLIC PWM_CONTROL,SP,CR,LF
000A                =1 63
001C                =1 64  RSEG at LCH
001C                =1 65
001C                =1 66  AX:      DSW      1          ; Temp registers used in conformance
001E                =1 67  DX:      DSW      1          ; with PIM-96(tm) conventions.
0020                =1 68  BX:      DSW      1
0022                =1 69  CX:      DSW      1
0022                =1 70
001C                =1 71  AL       EQU      AX       :BYTE
001D                =1 72  AH       EQU      (AX+1)    :BYTE
0020                =1 73  BL       EQU      BX       :BYTE
0020                =1 74
0020                =1 75  public ax, bx, cx, dx, al, ah, bl
0020                =1 76
0020                =1 77  $list      ; Turn listing back on
0020                =1 78  ; End of include file
0020                =1 79
0020                =1 80  ; A2D UTILITY COMMANDS/RESPONSES FOR "CONTROL_A2D"
0020                =1 81
0007                =1 82  busy      equ      7
0010                =1 83  con_b0     equ      00010000b;convert to BUFF0
0028                =1 84  dump_b0_p_s equ      00101000b; download BUFF0 as PAIRED SIGNED data
0028                =1 85  ;
0028                =1 86
0001                =1 87  AVR_NUM    equ      1          ; Number of times to average the waveform
0001                =1 88  ; AVR_NUM < 256

```

270189-46

MCS-96 MACRO ASSEMBLER FFT_MAIN_APNOTE

02/18/86

PAGE 3

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		89	
	0000	90	SCALE_FACTOR equ 0 ; Number of rights shifts performed on
		91	; output of FFT. Used to prevent overflow
		92	; on summation
		93	
	0100	94	PLOT_RES equ 256 ; Number of input units per plot unit
	0080	95	PLOT_RES_2 equ plot_res/2
	0100	96	PLOT_MAX equ plot_res+145 ; 145 chrs/row
		97	
		98	PUBLIC scale_factor, plot_res, plot_res_2, plot_max
		99	
		100	
	0024	101	OSEG at 24H ; common oseg area
		102	
	0024	103	tmpreal: dsl 1
	0028	104	tmpimag: dsl 1
	002C	105	wndptr: dsw 1
	002E	106	varptr: dsw 1
		107	
	0000	108	RSEG
	0000	109	fft_mode: dsb 1
	0001	110	error: dsb 1
	0002	111	avr_cnt: dsb 1
		112	PUBLIC error, fft_mode
		113	
		114	EXTRN sample_period, control_a2d
		115	
		116	
	0080	117	DSEG at 80h
	0080	118	XREAL: ; For FFT routine
	0080	119	DEST_BUFF_BASE: DSW 64 ; For A2D routine
	00C0	120	XIMAG equ XREAL+64 ; For FFT routine
		121	
		122	PUBLIC DEST_BUFF_BASE, XREAL, XIMAG
		123	
		124	
	0200	125	DSEG AT 200H
		126	
	0200	127	PLOT_IN:
	0200	128	FFT_OUT: DSW 32 ; For FFT routine
	0240	129	BUFF0_BASE: DSW 64 ; For A2D routine
	02C0	130	BUFF1_BASE: DSW 64 ; For A2D routine
		131	
		132	PUBLIC BUFF0_BASE, BUFF1_BASE, FFT_OUT, PLOT_IN
		133	\$eject

270189-47

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			134	
	2080		135	CSEG AT 2080H
			136	
			137	EXTERN INIT_OUTPUT, DRAW_GRAPH, CON_OUT ; For Plot Routine
			138	EXTERN FFT_CALC ; For FFT routine
			139	EXTERN A2D_BUFF_UTIL ; For A2D routine
			140	
	2080 A1000018	R	141	LD SP,#STACK
	2084 A30100301C		142	LD AX,3000H
	2089		143	SBE_WAIT:
	2089 E01CFD		144	djnz al,sbe_wait ; WAIT FOR SBE TO CLEAR SERIAL PORT INTERRUPTS
	208C E01DFA		145	djnz ah,sbe_wait
			146	
	208F EF0000	E	147	BEGIN: CALL INIT_OUTPUT ; Initialize serial port
			148	
	2092		149	NEW_TRANSFORM_SET:
	2092 B10000	R	150	ldb fft_mode,#0000B ; Bit 0 - Real data / Tabled data#
			151	; Bit 1 - Windowed / Unwindowed#
			152	; Bit 2 - 10log Mag^2 / Magnitude#
			153	; Bit 3 - 256db plot / Normal Plot#
	2095 B10102	R	154	ldb avr_cnt,#avr_num
	2098 0120		155	clr bx
	209A C321000200		156	CLRRAM: st zero,fft_out[bx] ; clear fft magnitude array
	209F 65020020		157	add bx,#2
	20A3 89400020		158	cmp bx,#64
	20A7 DF01		159	blt CLRRAM
			160	
	20A9 300004	R	161	C_load: bbc fft_mode,0,do_tab ; Branch if real data is not used
	20AC 2819		162	CALL LOAD_DATA
	20AE 2002		163	br C_win
			164	
	20B0 282F		165	do_tab: CALL TABLE_LOAD
			166	
	20B2 310002	R	167	C_win: bbc fft_mode,1,calc ; Branch if windowing is not used
	20B5 28CB		168	CALL DO_WINDOW
			169	
	20B7 EF0000	E	170	CALC: CALL FFT_CALC
	20BA 980001	R	171	errtrp: cmpb error,zero
	20BD D7FB		172	jne errtrp
			173	
	20BF E00205	R	174	DJNZ avr_cnt, LOAD_DATA ; repeat for AVR_NUM counts
			175	
	20C2 EF0000	E	176	CALL DRAW_GRAPH
			177	
	20C5 27CB		178	BR NEW_TRANSFORM_SET
			179	\$eject

MCS-96 MACRO ASSEMBLER FFT_MAIN_APNOTE

02/18/86

PAGE 5

```

ERR LOC OBJECT          LINE      SOURCE STATEMENT
                                180
                                ;
20C7                   181      LOAD_DATA:          ;;;;  LOAD DATA INTO RAM
                                182
20C7 B1000F            183          ldb      port1,$00          ;**** FOR INDICATION ONLY
                                184
20CA                   185      SET_A2D:
20CA B11000            E 186          ldb      control_a2d,$con_b0      ; Set converter for buffer0
20C9 310100            E 187          orb      control_a2d,$01      ; Convert channel i
20D0 A1320000          E 188          ld       sample_period,$50      ; 100 us sample period
                                189
20D4 EF0000            E 190          CALL     a2d_buff_util      ; Start the conversion process
20D7 3F00FD            E 191          jbs      control_a2d,busy,$      ; wait for all conversions to be done
                                192
20DA                   193      Down_load:
20DA B12800            E 194          ldb      control_a2d,$dump_b0_p_s      ; download b0 paired/signed
20DD EF0000            E 195          CALL     a2d_buff_util
20E0 F0                196          RET
                                197
                                198
                                199      TABLE_LOAD:
20E1 0120              200          clr      bx
20E3 A102211C          201          ld      ax,$DATA0      ; Load tabled data for testing
20E7 A21D22            202      load:      ld      cx,[ax]+755
20EA A21D1E            203          ld      dx,[ax]+
20ED C321800022        204          st      cx,xreal[bx]
20F2 C321C0001E        205          st      dx,ximag[bx]
20F7 65020020          206          add     bx,$2
20FB 89400020          207          cmp     bx,$64
20FF DEE6              208          blt     LOAD
2101 F0                209          RET
                                210
2102 0100              211      DATA0:      ; SQUARE WAVE
                                212
2102 FF7FFF7FFF7FFF7F  213      DCW      32767, 32767, 32767, 32767, 32767, 32767, 32767, 32767
2112 FF7FFF7FFF7FFF7F  214      DCW      32767, 32767, 32767, 32767, 32767, 32767, 32767, 32767
2122 FF7FFF7FFF7FFF7F  215      DCW      32767, 32767, 32767, 32767, 32767, 32767, 32767, 32767
2132 FF7FFF7FFF7FFF7F  216      DCW      32767, 32767, 32767, 32767, 32767, 32767, 32767, 32767
2142 0180018001800180  217      DCW      -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
2152 0180018001800180  218      DCW      -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
2162 0180018001800180  219      DCW      -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
2172 0180018001800180  220      DCW      -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
                                221
2172 0180018001800180  222      $object
                                223
                                224
                                225
                                226
                                227
                                228
                                229
                                230
                                231
                                232
                                233
                                234
                                235
                                236
                                237
                                238
                                239
                                240
                                241
                                242
                                243
                                244
                                245
                                246
                                247
                                248
                                249
                                250
                                251
                                252
                                253
                                254
                                255
                                256
                                257
                                258
                                259
                                260
                                261
                                262
                                263
                                264
                                265
                                266
                                267
                                268
                                269
                                270
                                271
                                272
                                273
                                274
                                275
                                276
                                277
                                278
                                279
                                280
                                281
                                282
                                283
                                284
                                285
                                286
                                287
                                288
                                289
                                290
                                291
                                292
                                293
                                294
                                295
                                296
                                297
                                298
                                299
                                300
                                301
                                302
                                303
                                304
                                305
                                306
                                307
                                308
                                309
                                310
                                311
                                312
                                313
                                314
                                315
                                316
                                317
                                318
                                319
                                320
                                321
                                322
                                323
                                324
                                325
                                326
                                327
                                328
                                329
                                330
                                331
                                332
                                333
                                334
                                335
                                336
                                337
                                338
                                339
                                340
                                341
                                342
                                343
                                344
                                345
                                346
                                347
                                348
                                349
                                350
                                351
                                352
                                353
                                354
                                355
                                356
                                357
                                358
                                359
                                360
                                361
                                362
                                363
                                364
                                365
                                366
                                367
                                368
                                369
                                370
                                371
                                372
                                373
                                374
                                375
                                376
                                377
                                378
                                379
                                380
                                381
                                382
                                383
                                384
                                385
                                386
                                387
                                388
                                389
                                390
                                391
                                392
                                393
                                394
                                395
                                396
                                397
                                398
                                399
                                400
                                401
                                402
                                403
                                404
                                405
                                406
                                407
                                408
                                409
                                410
                                411
                                412
                                413
                                414
                                415
                                416
                                417
                                418
                                419
                                420
                                421
                                422
                                423
                                424
                                425
                                426
                                427
                                428
                                429
                                430
                                431
                                432
                                433
                                434
                                435
                                436
                                437
                                438
                                439
                                440
                                441
                                442
                                443
                                444
                                445
                                446
                                447
                                448
                                449
                                450
                                451
                                452
                                453
                                454
                                455
                                456
                                457
                                458
                                459
                                460
                                461
                                462
                                463
                                464
                                465
                                466
                                467
                                468
                                469
                                470
                                471
                                472
                                473
                                474
                                475
                                476
                                477
                                478
                                479
                                480
                                481
                                482
                                483
                                484
                                485
                                486
                                487
                                488
                                489
                                490
                                491
                                492
                                493
                                494
                                495
                                496
                                497
                                498
                                499
                                500
                                501
                                502
                                503
                                504
                                505
                                506
                                507
                                508
                                509
                                510
                                511
                                512
                                513
                                514
                                515
                                516
                                517
                                518
                                519
                                520
                                521
                                522
                                523
                                524
                                525
                                526
                                527
                                528
                                529
                                530
                                531
                                532
                                533
                                534
                                535
                                536
                                537
                                538
                                539
                                540
                                541
                                542
                                543
                                544
                                545
                                546
                                547
                                548
                                549
                                550
                                551
                                552
                                553
                                554
                                555
                                556
                                557
                                558
                                559
                                560
                                561
                                562
                                563
                                564
                                565
                                566
                                567
                                568
                                569
                                570
                                571
                                572
                                573
                                574
                                575
                                576
                                577
                                578
                                579
                                580
                                581
                                582
                                583
                                584
                                585
                                586
                                587
                                588
                                589
                                590
                                591
                                592
                                593
                                594
                                595
                                596
                                597
                                598
                                599
                                600
                                601
                                602
                                603
                                604
                                605
                                606
                                607
                                608
                                609
                                610
                                611
                                612
                                613
                                614
                                615
                                616
                                617
                                618
                                619
                                620
                                621
                                622
                                623
                                624
                                625
                                626
                                627
                                628
                                629
                                630
                                631
                                632
                                633
                                634
                                635
                                636
                                637
                                638
                                639
                                640
                                641
                                642
                                643
                                644
                                645
                                646
                                647
                                648
                                649
                                650
                                651
                                652
                                653
                                654
                                655
                                656
                                657
                                658
                                659
                                660
                                661
                                662
                                663
                                664
                                665
                                666
                                667
                                668
                                669
                                670
                                671
                                672
                                673
                                674
                                675
                                676
                                677
                                678
                                679
                                680
                                681
                                682
                                683
                                684
                                685
                                686
                                687
                                688
                                689
                                690
                                691
                                692
                                693
                                694
                                695
                                696
                                697
                                698
                                699
                                700
                                701
                                702
                                703
                                704
                                705
                                706
                                707
                                708
                                709
                                710
                                711
                                712
                                713
                                714
                                715
                                716
                                717
                                718
                                719
                                720
                                721
                                722
                                723
                                724
                                725
                                726
                                727
                                728
                                729
                                730
                                731
                                732
                                733
                                734
                                735
                                736
                                737
                                738
                                739
                                740
                                741
                                742
                                743
                                744
                                745
                                746
                                747
                                748
                                749
                                750
                                751
                                752
                                753
                                754
                                755
                                756
                                757
                                758
                                759
                                760
                                761
                                762
                                763
                                764
                                765
                                766
                                767
                                768
                                769
                                770
                                771
                                772
                                773
                                774
                                775
                                776
                                777
                                778
                                779
                                780
                                781
                                782
                                783
                                784
                                785
                                786
                                787
                                788
                                789
                                790
                                791
                                792
                                793
                                794
                                795
                                796
                                797
                                798
                                799
                                800
                                801
                                802
                                803
                                804
                                805
                                806
                                807
                                808
                                809
                                810
                                811
                                812
                                813
                                814
                                815
                                816
                                817
                                818
                                819
                                820
                                821
                                822
                                823
                                824
                                825
                                826
                                827
                                828
                                829
                                830
                                831
                                832
                                833
                                834
                                835
                                836
                                837
                                838
                                839
                                840
                                841
                                842
                                843
                                844
                                845
                                846
                                847
                                848
                                849
                                850
                                851
                                852
                                853
                                854
                                855
                                856
                                857
                                858
                                859
                                860
                                861
                                862
                                863
                                864
                                865
                                866
                                867
                                868
                                869
                                870
                                871
                                872
                                873
                                874
                                875
                                876
                                877
                                878
                                879
                                880
                                881
                                882
                                883
                                884
                                885
                                886
                                887
                                888
                                889
                                890
                                891
                                892
                                893
                                894
                                895
                                896
                                897
                                898
                                899
                                900
                                901
                                902
                                903
                                904
                                905
                                906
                                907
                                908
                                909
                                910
                                911
                                912
                                913
                                914
                                915
                                916
                                917
                                918
                                919
                                920
                                921
                                922
                                923
                                924
                                925
                                926
                                927
                                928
                                929
                                930
                                931
                                932
                                933
                                934
                                935
                                936
                                937
                                938
                                939
                                940
                                941
                                942
                                943
                                944
                                945
                                946
                                947
                                948
                                949
                                950
                                951
                                952
                                953
                                954
                                955
                                956
                                957
                                958
                                959
                                960
                                961
                                962
                                963
                                964
                                965
                                966
                                967
                                968
                                969
                                970
                                971
                                972
                                973
                                974
                                975
                                976
                                977
                                978
                                979
                                980
                                981
                                982
                                983
                                984
                                985
                                986
                                987
                                988
                                989
                                990
                                991
                                992
                                993
                                994
                                995
                                996
                                997
                                998
                                999
                                1000

```

270189-49

MCS-96 MACRO ASSEMBLER FFT_MAIN_APNOTE

02/18/86

PAGE 6

```

ERR LOC OBJECT          LINE      SOURCE STATEMENT
2182          223
2182 012C          224 DO_WINDOW:          ;;; PERFORM HANNING WINDOW
2184 012E          225         clr      wndptr
2186          226         clr      varptr          ; Windowing provides an effective
2186          227 WINDOW:          ; divide by 2 because of the multiply
2186 A32BE211C        228         ld      ax,hanning[wndptr]
2186 A32DC02120        229         ld      bx,hanning+2[wndptr]
2190 FE4F2F80001C24    230 DCN      mul      tmpreal,ax,xreal[varptr]
2197 FE4F2FC0002028    231 DCN      mul      tmpimag,bx,ximag[varptr]
219E 0D0124          232 DCN      shl     tmpreal,#1
21A1 0D0128          233 DCN      shl     tmpimag,#1          ; Compensate for the divide by 2
21A4 C32F800026        234 DCN      st      tmpreal+2,xreal[varptr]
21A9 C32FC0002A        235 DCN      st      tmpimag+2,ximag[varptr]
21AE 6504002C          236 DCN      add     wndptr,#4
21B2 6502002E          237 DCN      add     varptr,#2
21B6 8940002E          238 DCN      cmp     varptr,#64
21BA D7CA            239 DCN      jne     window
21BC F0              240 RST
21BE          241
21BE          242 HANNING:          ; Windowing function
21BE          243
21BE 00004F003B01C102  244 DCW      0, 79, 315, 705, 1247, 1935, 2761, 3719
21CE BF126617711CD421  245 DCW      4799, 5990, 7281, 8660, 10114, 11628, 13187, 14778
21DE 004045467C4C9352  246 DCW      16384, 17989, 19580, 21139, 22653, 24107, 25486, 26777
21EE 406D757136757078  247 DCW      27968, 29048, 30006, 30832, 31520, 32062, 32452, 32688
21FE FF7FB07FC47E3E7D  248 DCW      32767, 32688, 32452, 32062, 31520, 30832, 30006, 29048
220E 406D99688E632B5E  249 DCW      27968, 26777, 25486, 24107, 22653, 21139, 19580, 17989
221E 0040BA3983336C2D  250 DCW      16384, 14778, 13187, 11628, 10114, 8660, 7281, 5990
222E BF12870EC90A8F07  251 DCW      4799, 3719, 2761, 1935, 1247, 705, 315, 79
223E 0000            252 DCW      0
2182          253
2182          254 $reject

```

270189-50

MCS-96 MACRO ASSEMBLER FFT_MAIN_APNOTE

02/18/86

PAGE 7

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
	3D00		255	
			256	CSRG AT 3D00H ; ADDITIONAL TABLES FOR TESTING
			257	; SINE 7.0 X
	3D00		258	DATA1:
	3D00	00003351897DE270	259	DCW 0, 20787, 32137, 28898, 12539, -9512, -27245, -32609
	3D10	7EA574F31C477C7A	260	DCW -23170, -3212, 18204, 31356, 30273, 15446, -6393, -25329
	3D20	01800F9D08E7563C	261	DCW -32767, -25329, -6392, 15446, 30273, 31356, 18204, -3212
	3D30	7EA574F31C477C7A	262	DCW -23170, -3212, 18204, 31356, 30273, 15446, -6393, -25329
	3D40	0000CDAE77821E8F	263	DCW -0, -20787, -32137, -28898, -12539, 9512, 27245, 32609
	3D50	825A80C84B88485	264	DCW 23170, 3212, -18204, -31356, -30273, -15446, 6393, 25329
	3D60	FF7FF162F818AAC3	265	DCW 32767, 25329, 6392, -15446, -30273, -31356, -18204, 3212
	3D70	825A617F6D6A2825	266	DCW 23170, 32609, 27245, 9512, -12539, -28898, -32137, -20787
			267	
	3D80		268	DATA2: ; SINE 7.5 X
			269	
	3D80	0000F555617FCF66	270	DCW 0, 22005, 32609, 26319, 6393, -16846, -31356, -29621
	3D90	05CF1F2BE270297C	271	DCW -12539, 11039, 28898, 31785, 18204, -4808, -25329, -32728
	3DA0	7EA5B8F933519C7E	272	DCW -23170, -1608, 20787, 32412, 27245, 7962, -15446, -30852
	3DB0	BF8946C92825C96D	273	DCW -30273, -14010, 9512, 28105, 32137, 19519, -3212, -24279
	3DC0	018029A174F33F4C	274	DCW -32767, -24279, -3212, 19519, 32137, 28105, 9512, -14010
	3DD0	BF897C87AAC31A1F	275	DCW -30273, -30852, -15446, 7962, 27245, 32412, 20787, -1608
	3DE0	7EA52880F9D3BED	276	DCW -23170, -32728, -25329, -4808, 18205, 31785, 28898, 11039
	3DF0	05CF4B8C848533BE	277	DCW -12539, -29621, -31356, -16845, 6393, 26319, 32609, 22005
			278	
	3E00		279	DATA3: ; .707*SINE 7.5X
			280	
	3E00	0000C63C0F5AAF48	281	DCW 0, 15558, 23055, 18607, 4520, -11910, -22169, -20942
	3E10	5FDD7C18CF4FC857	282	DCW -8865, 7804, 20431, 22472, 12870, -3399, -17908, -23138
	3E20	03C08FFB69398459	283	DCW -16381, -1137, 14697, 22916, 19262, 5629, -10921, -21812
	3E30	65AC4FD9451A9E4D	284	DCW -21403, -9905, 6725, 19870, 22721, 13800, -2271, -17165
	3E40	82A5F3BC21F7E835	285	DCW -23166, -17165, -2271, 13800, 22721, 19870, 6725, -9905
	3E50	65ACCCAA58D5FD15	286	DCW -21403, -21812, -10920, 5629, 19262, 22916, 14696, -1137
	3E60	03C09EA50CBAB9F2	287	DCW -16381, -23138, -17908, -3399, 12871, 22472, 20431, 7804
	3E70	5FDD32AE67A97AD1	288	DCW -8865, -20942, -22169, -11910, 4520, 18607, 23055, 15557
			289	
	3E80		290	DATA4: ; .707*SINE(11X) /16
			291	
	3E80	0000FD04B40472FF	292	DCW 0, 1277, 1204, -142, -1338, -1119, 282, 1386
	3E90	00045CFE74FA69FC	293	DCW 1024, -420, -1420, -919, 554, 1441, 804, -683
	3EA0	58FA55FD2403A105	294	DCW -1448, -683, 804, 1441, 554, -919, -1420, -420
	3EB0	00046A051A01A1FB	295	DCW 1024, 1386, 282, -1119, -1338, -142, 1204, 1277
	3EC0	000003FB4CFB8E00	296	DCW -0, -1277, -1204, 142, 1338, 1119, -282, -1386
	3ED0	00FCA4018C059703	297	DCW -1024, 420, 1420, 919, -554, -1441, -804, 683
	3EE0	A805AB02DCFC5FFA	298	DCW 1448, 683, -804, -1441, -554, 919, 1420, 420
	3EF0	00FC9FAE6FE5F04	299	DCW -1024, -1386, -282, 1119, 1338, 142, -1204, -1277
			300	
	3F00		301	DATA5: ; .707*(SINE 7.5X + 1/16 SINE 11X)

270189-51

MCS-96 MACRO ASSEMBLER FFT_MAIN_APNOTE

02/18/86

PAGE 8

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		302	
3F00	0000C241C35E2148	303	DCW 0, 16834, 24259, 18465, 3182, -13029, -21886, -19557
3F10	5EE1D81C434A3154	304	DCW -7842, 7384, 19011, 21553, 13425, -1958, -17103, -23821
3F20	5BB85F88D3C245F	305	DCW -17829, -1819, 15501, 24356, 19816, 4710, -12341, -22232
3F30	65B0B9DE5F1B3F49	306	DCW -20379, -8519, 7007, 18751, 21383, 13658, -1067, -15888
3F40	82A5F6B76DF27636	307	DCW -23166, -18442, -3475, 13942, 24059, 20990, 6442, -11290
3F50	65A870ACE4DA9419	308	DCW -22427, -21392, -9500, 6548, 18708, 21475, 13892, -454
3F60	ABC548ABE8B618ED	309	DCW -14933, -22456, -18712, -4840, 12317, 23391, 21851, 8225
3F70	5FD9CBA84DASD9D5	310	DCW -9889, -22328, -22451, -10791, 5857, 18749, 21851, 14281
		311	
		312	DCW 0, 7341, 1804, -145, -1250, -1118, 328, 1286
3F80		313	END

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

270189-52

3F00	0000C241C35E2148	303	DCW 0, 16834, 24259, 18465, 3182, -13029, -21886, -19557
3F10	5EE1D81C434A3154	304	DCW -7842, 7384, 19011, 21553, 13425, -1958, -17103, -23821
3F20	5BB85F88D3C245F	305	DCW -17829, -1819, 15501, 24356, 19816, 4710, -12341, -22232
3F30	65B0B9DE5F1B3F49	306	DCW -20379, -8519, 7007, 18751, 21383, 13658, -1067, -15888
3F40	82A5F6B76DF27636	307	DCW -23166, -18442, -3475, 13942, 24059, 20990, 6442, -11290
3F50	65A870ACE4DA9419	308	DCW -22427, -21392, -9500, 6548, 18708, 21475, 13892, -454
3F60	ABC548ABE8B618ED	309	DCW -14933, -22456, -18712, -4840, 12317, 23391, 21851, 8225
3F70	5FD9CBA84DASD9D5	310	DCW -9889, -22328, -22451, -10791, 5857, 18749, 21851, 14281
		311	
		312	DCW 0, 7341, 1804, -145, -1250, -1118, 328, 1286
3F80		313	END

3F00	0000C241C35E2148	303	DCW 0, 16834, 24259, 18465, 3182, -13029, -21886, -19557
3F10	5EE1D81C434A3154	304	DCW -7842, 7384, 19011, 21553, 13425, -1958, -17103, -23821
3F20	5BB85F88D3C245F	305	DCW -17829, -1819, 15501, 24356, 19816, 4710, -12341, -22232
3F30	65B0B9DE5F1B3F49	306	DCW -20379, -8519, 7007, 18751, 21383, 13658, -1067, -15888
3F40	82A5F6B76DF27636	307	DCW -23166, -18442, -3475, 13942, 24059, 20990, 6442, -11290
3F50	65A870ACE4DA9419	308	DCW -22427, -21392, -9500, 6548, 18708, 21475, 13892, -454
3F60	ABC548ABE8B618ED	309	DCW -14933, -22456, -18712, -4840, 12317, 23391, 21851, 8225
3F70	5FD9CBA84DASD9D5	310	DCW -9889, -22328, -22451, -10791, 5857, 18749, 21851, 14281
		311	
		312	DCW 0, 7341, 1804, -145, -1250, -1118, 328, 1286
3F80		313	END

3F00	0000C241C35E2148	303	DCW 0, 16834, 24259, 18465, 3182, -13029, -21886, -19557
3F10	5EE1D81C434A3154	304	DCW -7842, 7384, 19011, 21553, 13425, -1958, -17103, -23821
3F20	5BB85F88D3C245F	305	DCW -17829, -1819, 15501, 24356, 19816, 4710, -12341, -22232
3F30	65B0B9DE5F1B3F49	306	DCW -20379, -8519, 7007, 18751, 21383, 13658, -1067, -15888
3F40	82A5F6B76DF27636	307	DCW -23166, -18442, -3475, 13942, 24059, 20990, 6442, -11290
3F50	65A870ACE4DA9419	308	DCW -22427, -21392, -9500, 6548, 18708, 21475, 13892, -454
3F60	ABC548ABE8B618ED	309	DCW -14933, -22456, -18712, -4840, 12317, 23391, 21851, 8225
3F70	5FD9CBA84DASD9D5	310	DCW -9889, -22328, -22451, -10791, 5857, 18749, 21851, 14281
		311	
		312	DCW 0, 7341, 1804, -145, -1250, -1118, 328, 1286
3F80		313	END

3F00	0000C241C35E2148	303	DCW 0, 16834, 24259, 18465, 3182, -13029, -21886, -19557
3F10	5EE1D81C434A3154	304	DCW -7842, 7384, 19011, 21553, 13425, -1958, -17103, -23821
3F20	5BB85F88D3C245F	305	DCW -17829, -1819, 15501, 24356, 19816, 4710, -12341, -22232
3F30	65B0B9DE5F1B3F49	306	DCW -20379, -8519, 7007, 18751, 21383, 13658, -1067, -15888
3F40	82A5F6B76DF27636	307	DCW -23166, -18442, -3475, 13942, 24059, 20990, 6442, -11290
3F50	65A870ACE4DA9419	308	DCW -22427, -21392, -9500, 6548, 18708, 21475, 13892, -454
3F60	ABC548ABE8B618ED	309	DCW -14933, -22456, -18712, -4840, 12317, 23391, 21851, 8225
3F70	5FD9CBA84DASD9D5	310	DCW -9889, -22328, -22451, -10791, 5857, 18749, 21851, 14281
		311	
		312	DCW 0, 7341, 1804, -145, -1250, -1118, 328, 1286
3F80		313	END

02/18/86

PAGE 8

SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation

INPUT FILES: :F2:FTMAIN.OBJ, :F2:FFTRUN.OBJ, :F2:PLOTSP.OBJ, :F2:A2DCON.OBJ
OUTPUT FILE: :F2:FFTOUT
CONTROLS SPECIFIED IN INVOCATION COMMAND:

INPUT MODULES INCLUDED:
:F2:FTMAIN.OBJ(FFT_MAIN_APNOTE) 02/18/86
:F2:FFTRUN.OBJ(FFT_RUN) 02/18/86
:F2:PLOTSP.OBJ(PLOT_SERIAL) 02/18/86
:F2:A2DCON.OBJ(A2D_BUFFERING_UTILITY) 02/18/86

SEGMENT MAP FOR :F2:FFTOUT(FFT_MAIN_APNOTE):

TYPE	BASE	LENGTH	ALIGNMENT	MODULE NAME
**RESERVED*	0000H	001AH		
REG	001AH	0001H	BYTE	PLOT_SERIAL
*** GAP ***	001BH	0001H		
REG	001CH	0008H	ABSOLUTE	FFT_MAIN_APNOTE
OVERLY	0024H	00035H	ABSOLUTE	FFT_RUN
OVERLAP	0024H	0010H	ABSOLUTE	PLOT_SERIAL
OVERLAP	0024H	000CH	ABSOLUTE	FFT_MAIN_APNOTE
*** GAP ***	0059H	0001H		
OVERLY	005AH	0006H	WORD	A2D_BUFFERING_UTILITY
REG	0060H	000CH	WORD	A2D_BUFFERING_UTILITY
REG	006CH	0003H	BYTE	FFT_MAIN_APNOTE
*** GAP ***	006FH	0011H		
DATA	0080H	0080H	ABSOLUTE	FFT_MAIN_APNOTE
STACK	0100H	001EH	WORD	
DATA	011EH	0080H	WORD	FFT_RUN
*** GAP ***	015EH	0062H		
DATA	0200H	0140H	ABSOLUTE	FFT_MAIN_APNOTE
*** GAP ***	0340H	1CC2H		
CODE	2002H	0002H	ABSOLUTE	A2D_BUFFERING_UTILITY
*** GAP ***	2004H	007CH		
CODE	2080H	01C0H	ABSOLUTE	FFT_MAIN_APNOTE
*** GAP ***	2240H	0040H		
CODE	2280H	0215H	ABSOLUTE	FFT_RUN
*** GAP ***	2495H	0068H		
CODE	2500H	0168H	ABSOLUTE	PLOT_SERIAL
CODE	2668H	00ECH	BYTE	A2D_BUFFERING_UTILITY
*** GAP ***	2754H	10ACH		
CODE	3800H	042AH	ABSOLUTE	FFT_RUN
*** GAP ***	3C2AH	00D6H		
CODE	3D00H	0280H	ABSOLUTE	FFT_MAIN_APNOTE
*** GAP ***	3F80H	C080H		

270189-53

6

Listing 3—Main Routine (Continued)

Several constants are then setup for other routines. The purpose of centrally locating these constants was the ease of modifying the operation of the routines. Note that `AVR_NUM` and `SCALE_FACTOR` must be changed at the same time. `SCALE_FACTOR` is the shift count used to divide each FFT output value before it is added to the output array. `AVR_NUM` must be less than $2^{**SCALE_FACTOR}$ or an overflow could occur. Next, the public variables are declared for the arrays and a few other parameters.

The program then begins by setting the stack pointer and waiting for the SBE-96 to finish talking to the terminal. If this is not done, there may be serial port interrupts occurring for the first twenty five milliseconds of program operation.

Initialization of the plotter is next, followed by setting the `FFT_MODE` byte. This byte controls the graphing, loading and magnitude calculation of the FFT data. Since `FFT_MODE` is declared PUBLIC in this module, and EXTERNAL in the PLOT module and FFTRUN module, the extra bits available in this byte can be used for future enhancements.

The next step is to clear the FFT output array. Since the FFT program can be set to average its results by dividing the output before adding it to the magnitude array, the array must be cleared before beginning the program.

Data is then loaded into the FFT input array by the code at `LOAD_DATA`, or the code at `TABLE_LOAD`, depending on the value of `FFT_MODE` bit 0. The tabled data located at `DATA0` is a square wave of magnitude 1. This waveform provides a reasonable test of the FFT algorithm, as many harmonics are generated. The results are also easy to check as the pattern contains half zeros, imaginary values which are always the same, and real values which decrease. Figure 13 shows the output in fractions, hexadecimal and decimal. The hexadecimal and decimal values are based on an output of 16384 being equal to 1.00.

Note that the magnitude is

$$\text{SQR}(\text{REAL}^2 + \text{IMAG}^2)$$

and the dB value is

$$10 \text{ LOG} ((\text{REAL}^2 + \text{IMAG}^2)/65536)$$

The divide by 65536 is used for the dB scale to provide a reasonable range for calculations. If this was not done, a 32-bit LOG function would have been needed.

After the data is loaded, the data is optionally windowed, based on `FFT_MODE` bit 1, and the FFT program is called. Once the loop has been performed `AVR_CNT` times, the graph is drawn by the plot routine.

Appended to the main routine is the `FFTOUT.M96` Listing. This is provided by the relocater and linker, RL96. With this listing and the main program, it is possible to determine which sections of code are at which addresses.

Using the modular programming methods employed here, it is reasonably easy to debug code. By emulating the program in a relatively high level language, each routine can be checked for functionality against a known standard. The closer the high level implementation matches the ASM96 version, the more possible checkpoints there are between the two routines.

Once all of the program routines (modules) can be shown to work individually, the main program should work unless there is unwanted interaction between the modules. These interactions can be checked by verifying the inputs and outputs of each module. The assembly language locations to perform the program breaks can be retrieved by absolutely locating the main module. The other modules can be dynamically located by RL96.

The more interactive program modules are, the more difficult the program becomes to debug. This is especially true when multiple interrupts are occurring, and several of the interrupt routines are themselves interruptable. In these cases, it may be necessary to use debugging equipment with trace capability, like the VLSiCE-96. If this type of equipment is not available, then using I/O ports to indicate the entering and leaving of each routine may be useful. In this way it will be possible to watch the action of the program on an oscilloscope or logic analyzer. There are several places within this code that I/O port toggling has been used as an aid to debugging the program. These lines of code are marked "FOR INDICATION ONLY."

K	Fractional			dB	Decimal			Hexadecimal		
	REAL	IMAG	MAG ²		REAL	IMAG	MAG ²	REAL	IMAG	MAG ²
0	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
1	0.0625	-1.2722	1.2738	38.225	1024	-20843	20868	400	AE95	5184
2	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
3	0.0625	-0.4213	0.4260	28.710	1024	-6903	6978	400	E509	1B42
4	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
5	0.0625	-0.2495	0.2572	24.329	1024	-4088	4214	400	F008	1076
6	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
7	0.0625	-0.1747	0.1855	21.491	1024	-2862	3039	400	F4D2	BDF
8	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
9	0.0625	-0.1321	0.1462	19.421	1024	-2165	2395	400	F78B	95B
10	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
11	0.0625	-0.1043	0.1216	17.820	1024	-1708	1992	400	F954	7C8
12	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
13	0.0625	-0.0843	0.1049	16.540	1024	-1381	1719	400	FA9B	6B7
14	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
15	0.0625	-0.0690	0.0931	15.499	1024	-1130	1525	400	FB96	5F5
16	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
17	0.0625	-0.0566	0.0844	14.645	1024	-928	1382	400	FC60	566
18	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
19	0.0625	-0.0464	0.0778	13.944	1024	-759	1275	400	FD09	4FB
20	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
21	0.0625	-0.0375	0.0729	13.374	1024	-614	1194	400	FD9A	4AA
22	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
23	0.0625	-0.0296	0.0691	12.918	1024	-484	1133	400	FE1C	46D
24	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
25	0.0625	-0.0224	0.0664	12.564	1024	-366	1088	400	FE92	440
26	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
27	0.0625	-0.0157	0.0644	12.305	1024	-256	1056	400	FF00	420
28	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
29	0.0625	-0.0093	0.0632	12.135	1024	-152	1035	400	FF68	40B
30	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
31	0.0625	-0.0031	0.0626	12.051	1024	-50	1025	400	FFCE	401

Figure 13. FFT Output for a Square Wave Input

9.0 ANALOG TO DIGITAL CONVERTER MODULE

The module presented in Listing 4 is a general purpose one which converts analog values under interrupt control and stores them in one of two buffers. These buffers

can then be downloaded to another buffer, such as the input buffer to the FFT program. During downloading, this module can convert the data into signed or unsigned formats, and fill a linear or a paired array. A paired array is like the one used in the FFT transform program. It requires N data points placed alternately in two arrays, one starting at zero and the other at N/2.

MCS-96 MACRO ASSEMBLER A2D_BUFFERING UTILITY

02/18/86

PAGE 1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:A2DCON.A96

OBJECT FILE: :F2:A2DCON.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC OBJECT

LINE

SOURCE STATEMENT

\$pagelength(50)

A2D_Buffering_Utility module stacksize(12)

; Intel Corporation, July 16, 1985

; by Dave Ryan, Intel Applications Engineer

; This utility fills a memory buffer with A/D conversion results. The conversions are done under interrupt control, and are initiated when A2D_BUFF Util is called. The results of the conversions are placed in one of two buffers, called BUFF0 and BUFF1.

; This utility provides options for the selection of the buffer lengths, data format, sample period, conversion channel and time base. The utility also has a download routine that will load either buffer into a register file buffer. Output formats can also be chosen for the downloaded buffer. The data can be formatted as signed or unsigned linear or paired arrays.

; RUN-TIME OPTIONS

; Rather than use the STACK to pass controls, this utility gets its directions from 2 control words in memory. The utility expects that its control words are valid at the time A2D_BUFF Util is called and remain valid throughout A/D interrupt executions and downloads. The control words are:

Sample_Period ; WORD ; The time between samples in timer counts where the timer used has been specified

Control_A2D ; BYTE ; Control information for the utility: BITS

; 0-2 ; Channel Number
; 3 ; Signed Result/Unsigned Result#
; 4 ; Convert/Download#
; 5 ; BUFF1/BUFF0# for conversions
; BUFF0/BUFF1# for downloads
; 6 ; Linear/Paired#
; 7 ; Converter BUSY/IDLE#

; \$EJECT

270189-54

MCS-96 MACRO ASSEMBLER A2D_BUFFERING_UTILITY

02/18/86

PAGE 2

ERR LOC OBJECT

LINE SOURCE STATEMENT

```

42 ;
43 ; The following is a table of equates that can be used to simplify the
44 ; bit diddling requirements. If you are not running conversions concurrently
45 ; with downloads, always LDB Control_A2D with the following command then
46 ; ORB Control_A2D with the channel number you wish to convert if you are
47 ; starting a conversion.
48 ;
49 ; Once the utility is called, care must be taken when Control_A2d is
50 ; modified. You can cause downloads to occur while conversions are running,
51 ; but you cannot start conversions during a download. To do this, ORB to the
52 ; control byte with the appropriate bits set. Do NOT change the BUFF bit or
53 ; the BUSY bit. Just set the download bit and set the data format bits to the
54 ; correct values.
55 ;
56 ; The BUFF bit has opposite definitions for conversions and downloads. This
57 ; allows conversions to be done into BUFF0 while downloads come from BUFF1, and
58 ; vice versa.
59 ;
60 ; A2D UTILITY COMMANDS
61 ;
62 ; con_b0 equ 00010000b; convert to BUFF0
63 ; con_b1 equ 00110000b; " BUFF1
64 ;
65 ; dump_b0_l_u equ 01100000b; download BUFF0 as LINEAR UNSIGNED data
66 ; dump_b1_l_u equ 01000000b; " BUFF1 " " " "
67 ; dump_b0_p_u equ 00100000b; " BUFF0 " PAIRED " " "
68 ; dump_b1_p_u equ 00000000b; " BUFF1 " " " "
69 ; dump_b0_l_s equ 01101000b; download BUFF0 as LINEAR SIGNED data
70 ; dump_b1_l_s equ 01001000b; " BUFF1 " " " "
71 ; dump_b0_p_s equ 00101000b; " BUFF0 " PAIRED " " "
72 ; dump_b1_p_s equ 00001000b; " BUFF1 " " " "
73 ;
74 $eject

```

270189-20

270189-55

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			75	;
			76	; ASSEMBLY-TIME OPTIONS
			77	;
			78	; The base addresses and length of each conversion buffer and the destination
			79	; buffer are DECLARED EXTRNal in this utility. Other options such as selection
			80	; of the timer used as a timebase, the length of the buffer, and the effective
			81	; number of bits in the reported result are set at assembly time through use
			82	; of EQUates in this module.
			83	;
			84	; The following parameters need to be provided at assembly or link time.
			85	; The buffer bases are declared EXTRNal by this utility, while the buffer
			86	; length shift count and HSO commands are EQUated.
			87	;
			88	BUFF0_BASE ; The starting address of BUFF0
			89	BUFF1_BASE ; The starting address of BUFF1
			90	DEST_BUFF_BASE ; The starting address of the download
			91	; target buffer.
			92	;
			93	BUFF_LENGTH ; The number of SAMPLES that each
			94	; buffer must hold. must be >1 and <256
			95	;
			96	Shift_count ; The number of times that the conversion result is
			97	; to be shifted right from its natural left justified
			98	; position. Setting a shift count greater than 6 will
			99	; result in lost bits to the right. Rounding is NOT
			100	; done.
			101	;
			102	CLOCK ; Specify as either TIMER1 or T2CLK. This is the
			103	; timebase used for conversions.
			104	;
			105	Samples are stored as words in the buffers. The program stores
			106	conversions linearly in BUFF0 and BUFF1, and linearly or paired in the
			107	destination buffer as selected. If the download is to be paired, the first
			108	sample is placed in location DEST_BUFF_BASE, the second sample is placed in
			109	location (DEST_BUFF_BASE + BUFF_LENGTH), the third in (DEST_BUFF_BASE + 2),
			110	the fourth in (DEST_BUFF_BASE + 2 + BUFF_LENGTH), etc.
			111	;
			112	\$object

270189-56

MCS-96 MACRO ASSEMBLER A2D_BUFFERING_UTILITY

02/18/86

PAGE 4

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			113	;
			114	;NOTES ON EXECUTION
			115	;
			116	; When a utility call directs the initiation of a set of A2D conversions, the
	0000		117	; first conversion is begun at approximately one sample time plus 50 state
	0000		118	; times from when the utility was called. This assumes that no interrupts are
			119	; present.
	0000		120	;
	0000		121	; The conversion busy bit is set approximately 50 state times after a call
	0000		122	; to the utility, if the convert bit was set in the A2D_Control byte. The
	0000		123	; busy bit is cleared after all conversion results have been stored in the
			124	; result buffer designated (BUFF0 or BUFF1).
	0000		125	;
	0000		126	; Take great care in modifying the A2D_Control byte to do a download while
	0000		127	; conversions are taking place. You can never download a buffer that is
	0000		128	; being converted into. The results would be invalid.
			129	\$object
	0000		130	;
			131	;
			132	;
			133	;
			134	;
			135	;
	0000		136	;
			137	;
	0000		138	;
			139	;
	0000		140	;
			141	;
	0000		142	;
			143	;
	0000		144	;
			145	;
	0000		146	;
			147	;
	0000		148	;
			149	;
	0000		150	;
			151	;
	0000		152	;
			153	;
	0000		154	;
			155	;
	0000		156	;
			157	;
	0000		158	;
			159	;
	0000		160	;
			161	;
	0000		162	;
			163	;
	0000		164	;
			165	;
	0000		166	;
			167	;
	0000		168	;
			169	;
	0000		170	;
			171	;
	0000		172	;
			173	;
	0000		174	;
			175	;
	0000		176	;
			177	;
	0000		178	;
			179	;
	0000		180	;
			181	;
	0000		182	;
			183	;
	0000		184	;
			185	;
	0000		186	;
			187	;
	0000		188	;
			189	;
	0000		190	;
			191	;
	0000		192	;
			193	;
	0000		194	;
			195	;
	0000		196	;
			197	;
	0000		198	;
			199	;
	0000		200	;
			201	;
	0000		202	;
			203	;
	0000		204	;
			205	;
	0000		206	;
			207	;
	0000		208	;
			209	;
	0000		210	;
			211	;
	0000		212	;
			213	;
	0000		214	;
			215	;
	0000		216	;
			217	;
	0000		218	;
			219	;
	0000		220	;
			221	;
	0000		222	;
			223	;
	0000		224	;
			225	;
	0000		226	;
			227	;
	0000		228	;
			229	;
	0000		230	;
			231	;
	0000		232	;
			233	;
	0000		234	;
			235	;
	0000		236	;
			237	;
	0000		238	;
			239	;
	0000		240	;
			241	;
	0000		242	;
			243	;
	0000		244	;
			245	;
	0000		246	;
			247	;
	0000		248	;
			249	;
	0000		250	;
			251	;
	0000		252	;
			253	;
	0000		254	;
			255	;
	0000		256	;
			257	;
	0000		258	;
			259	;
	0000		260	;
			261	;
	0000		262	;
			263	;
	0000		264	;
			265	;
	0000		266	;
			267	;
	0000		268	;
			269	;
	0000		270	;
			271	;
	0000		272	;
			273	;
	0000		274	;
			275	;
	0000		276	;
			277	;
	0000		278	;
			279	;
	0000		280	;
			281	;
	0000		282	;
			283	;
	0000		284	;
			285	;
	0000		286	;
			287	;
	0000		288	;
			289	;
	0000		290	;
			291	;
	0000		292	;
			293	;
	0000		294	;
			295	;
	0000		296	;
			297	;
	0000		298	;
			299	;
	0000		300	;
			301	;
	0000		302	;
			303	;
	0000		304	;
			305	;
	0000		306	;
			307	;
	0000		308	;
			309	;
	0000		310	;
			311	;
	0000		312	;
			313	;
	0000		314	;
			315	;
	0000		316	;
			317	;
	0000		318	;
			319	;
	0000		320	;
			321	;
	0000		322	;
			323	;
	0000		324	;
			325	;
	0000		326	;
			327	;
	0000		328	;
			329	;
	0000		330	;
			331	;
	0000		332	;
			333	;
	0000		334	;
			335	;
	0000		336	;
			337	;
	0000		338	;
			339	;
	0000		340	;
			341	;
	0000		342	;
			343	;
	0000		344	;
			345	;
	0000		346	;
			347	;
	0000		348	;
			349	;
	0000		350	;
			351	;
	0000		352	;
			353	;
	0000		354	;
			355	;
	0000		356	;
			357	;
	0000		358	;
			359	;
	0000		360	;
			361	;
	0000		362	;
			363	;
	0000		364	;
			365	;
	0000		366	;
			367	;
	0000		368	;
			369	;
	0000		370	;
			371	;
	0000		372	;
			373	;
	0000		374	;
			375	;
	0000		376	;
			377	;
	0000		378	;
			379	;
	0000		380	;
			381	;
	0000		382	;
			383	;
	0000		384	;
			385	;
	0000		386	;
			387	;
	0000		388	;
			389	;
	0000		390	;
			391	;
	0000		392	;
			393	;
	0000		394	;
			395	;
	0000		396	;
			397	;
	0000		398	;
			399	;
	0000		400	;
			401	;
	0000		402	;
			403	;
	0000		404	;
			405	;
	0000		406	;
			407	;
	0000		408	;
			409	;
	0000		410	;
			411	;
	0000		412	;
			413	;
	0000		414	

PAGE 5

270189-58

Listing 4—A to D Converter Routine (Continued)

6-152

PAGE 6

270189-59

Listing 4—A to D Converter Routine (Continued)

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			207	
	0000		208	A2D_BUFF_Util:
			209	
	0000 3C0962	R	210	JBS Control_A2D, Con_Dwn, Convert ; Select convert or download
	0003		211	Download:
	0003 A1000000	E	212	LD src_ptr,#BUFF1_BASE
	0007 350904	R	213	JBC Control_A2D, B0_B1, Set_Data_Format
			214	
	000A		215	Download_BUFF0:
	000A A1000000	E	216	LD src_ptr,#BUFF0_BASE
			217	
			218	
	000E		219	Set_Data_Format: ; Choose linear or paired
	000E A1000002	E	220	LD dest_ptr, #DEST_BUFF_BASE
	0012 B14004	R	221	LDB loop_count,#BUFF_LENGTH
	0015 3E091D	R	222	JBS Control_A2D, Lin_Par, Linear_data_loop
			223	
			224	
	0018 180104	R	225	PAIRED: SHRB loop_count,#1 ; The paired data routine uses 1/2
			226	; as many loops as the unpaired
			227	Paired_Data_loop:
	001B	R	228	LD adudtemp0,[src_ptr]+ ; Move even word
	001E C20200	R	229	ST adudtemp0,[dest_ptr]
	0021 65400002	R	230	ADD dest_ptr,#BUFF_LENGTH ; Length = # of words = 1/2 # of bytes
			231	
	0025 A20000	R	232	LD adudtemp0,[src_ptr]+ ; Move odd word
	0028 C20200	R	233	ST adudtemp0,[dest_ptr]+
	002B 69400002	R	234	SUB dest_ptr,#BUFF_LENGTH
			235	
	002F E004E9	R	236	DJNZ loop_count, Paired_Data_loop ; Loop until done
			237	
	0032 280D		238	CALL Convert_Data
	0034 F0		239	RET
			240	
			241	
	0035		242	Linear_Data_loop: ; Move data linearly
	0035 A20000	R	243	LD adudtemp0,[src_ptr]+
	0038 C20200	R	244	ST adudtemp0,[dest_ptr]+
			245	
	003B E004F7	R	246	DJNZ loop_count, Linear_Data_loop ; Loop until done
			247	
	003E 2801		248	CALL Convert_Data
	0040 F0		249	RET
			250	\$eject

270189-60

MCS-96 MACRO ASSEMBLER A2D_BUFFERING_UTILITY 02/18/86 PAGE 8

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		251	
		252	Convert_Data: ; Convert the data in the destination buffer
		253	
0041		254	LD loop_count,#BUFF_LENGTH
0045	A1000000	255	LD src_ptr,#DEST_BUFF_BASE
		256	
0049	A20000	257	Again: LD adudtemp0,[src_ptr]
004C	71C000	258	ANBR adudtemp0,#110000000h
004F	330909	259	JBC Control_A2D, DForm, Unsigned_Result
		260	
0052		261	Signed_Result:
0052	69E07F00	262	SUB adudtemp0,#7fe0h
0056	0A0100	263	SHRA adudtemp0,#Shift_Count
0059	2003	264	BR Replace_Sample
		265	
005B		266	Unsigned_Result:
005B	080100	267	SHR adudtemp0, #Shift_Count
		268	
005E		269	Replace_Sample:
005E	C20000	270	ST adudtemp0,[src_ptr]+
0061	E004E5	271	DJNZ loop_count,Again ; Loop until done
		272	
0064	F0	273	RET
		274	
0065		275	Convert: ;; Prepare to Start Conversions
		277	
0065	F2	278	PUSHF
		279	
0066	918009	280	ORB Control_A2D, #Busy ; set converter busy bit
		281	
0069	B13F08	282	LDB sample_count,#BUFF_LENGTH - 1
006C	A1000006	283	LD top_of_buffer,#BUFF0_BASE
0070	A1800004	284	aductemp1,#(BUFF0_BASE + 2*BUFF_LENGTH)
		285	
0074	350908	286	JBC Control_A2D, B0 B1, Start_Conversions
0077	A1000006	287	LD top_of_buffer,#BUFF1_BASE
007B	A1800004	288	aductemp1,#(BUFF1_BASE + 2*BUFF_LENGTH)
		289	\$eject

310198-85

270189-61

MCS-96 MACRO ASSEMBLER A2D_BUFFERING_UTILITY

02/18/86

PAGE 9

```

ERR LOC OBJECT          LINE    SOURCE STATEMENT
                                290
                                291      Start_Conversions:
                                292
007F          51070900    E      293      ANDB    ad_command,Control_A2D,$00000111b    ;load channel number
                                294
0083 440A0A02          R      295      ADD     aductemp0,CLOCK,Sample_Period    ;start first conversion
                                296      ;one sample time from
                                297      ;now
                                298
                                299      ;----- Load_HSO_Command Start_A2D ; Start A2D at Time=aductemp0
                                300      ;-----
008B 71000004          E      303      POP     temp    ; get a copy of the psw
008D CC00 0000          R      304      POP     temp    ; get a copy of the psw
008F 300000          E      305      ;----- Load_HSO_Command HSO_0_high ; set hso.0 high at conversion
                                306      ;-----
0093 71000004          E      310      ;----- ; start time for external S/H
                                311      ;-----
0095 81020200          R      312      OR      temp,$202h    ; enable a2d interrupts
                                313
0099 640A02          R      314      ADD     aductemp0,Sample_Period    ; set conversion period psw
                                315
                                316      Load_HSO_Command Start_A2D    ; start second conversion one
                                317      ; sample time from the first
                                318
                                319      ;-----
00A2 C800          R      321      ;----- ; put psw back on stack
                                322      PUSH    temp
                                323
                                324      Load_HSO_Command HSO_0_low    ;lower hso.0 for external S/H
                                325
                                326      ;-----
00AA F3          E      329      POPF    ; pop psw
00AB F0          E      330      RET     ; return
                                331      $eject    ;-----
                                332
                                333
                                334
                                335
                                336
                                337
                                338
                                339
                                340
                                341
                                342
                                343
                                344
                                345
                                346
                                347
                                348
                                349
                                350
                                351
                                352
                                353
                                354
                                355
                                356
                                357
                                358
                                359
                                360
                                361
                                362
                                363
                                364
                                365
                                366
                                367
                                368
                                369
                                370
                                371
                                372
                                373
                                374
                                375
                                376
                                377
                                378
                                379
                                380
                                381
                                382
                                383
                                384
                                385
                                386
                                387
                                388
                                389
                                390
                                391
                                392
                                393
                                394
                                395
                                396
                                397
                                398
                                399
                                400
                                401
                                402
                                403
                                404
                                405
                                406
                                407
                                408
                                409
                                410
                                411
                                412
                                413
                                414
                                415
                                416
                                417
                                418
                                419
                                420
                                421
                                422
                                423
                                424
                                425
                                426
                                427
                                428
                                429
                                430
                                431
                                432
                                433
                                434
                                435
                                436
                                437
                                438
                                439
                                440
                                441
                                442
                                443
                                444
                                445
                                446
                                447
                                448
                                449
                                450
                                451
                                452
                                453
                                454
                                455
                                456
                                457
                                458
                                459
                                460
                                461
                                462
                                463
                                464
                                465
                                466
                                467
                                468
                                469
                                470
                                471
                                472
                                473
                                474
                                475
                                476
                                477
                                478
                                479
                                480
                                481
                                482
                                483
                                484
                                485
                                486
                                487
                                488
                                489
                                490
                                491
                                492
                                493
                                494
                                495
                                496
                                497
                                498
                                499
                                500
                                501
                                502
                                503
                                504
                                505
                                506
                                507
                                508
                                509
                                510
                                511
                                512
                                513
                                514
                                515
                                516
                                517
                                518
                                519
                                520
                                521
                                522
                                523
                                524
                                525
                                526
                                527
                                528
                                529
                                530
                                531
                                532
                                533
                                534
                                535
                                536
                                537
                                538
                                539
                                540
                                541
                                542
                                543
                                544
                                545
                                546
                                547
                                548
                                549
                                550
                                551
                                552
                                553
                                554
                                555
                                556
                                557
                                558
                                559
                                560
                                561
                                562
                                563
                                564
                                565
                                566
                                567
                                568
                                569
                                570
                                571
                                572
                                573
                                574
                                575
                                576
                                577
                                578
                                579
                                580
                                581
                                582
                                583
                                584
                                585
                                586
                                587
                                588
                                589
                                590
                                591
                                592
                                593
                                594
                                595
                                596
                                597
                                598
                                599
                                600
                                601
                                602
                                603
                                604
                                605
                                606
                                607
                                608
                                609
                                610
                                611
                                612
                                613
                                614
                                615
                                616
                                617
                                618
                                619
                                620
                                621
                                622
                                623
                                624
                                625
                                626
                                627
                                628
                                629
                                630
                                631
                                632
                                633
                                634
                                635
                                636
                                637
                                638
                                639
                                640
                                641
                                642
                                643
                                644
                                645
                                646
                                647
                                648
                                649
                                650
                                651
                                652
                                653
                                654
                                655
                                656
                                657
                                658
                                659
                                660
                                661
                                662
                                663
                                664
                                665
                                666
                                667
                                668
                                669
                                670
                                671
                                672
                                673
                                674
                                675
                                676
                                677
                                678
                                679
                                680
                                681
                                682
                                683
                                684
                                685
                                686
                                687
                                688
                                689
                                690
                                691
                                692
                                693
                                694
                                695
                                696
                                697
                                698
                                699
                                700
                                701
                                702
                                703
                                704
                                705
                                706
                                707
                                708
                                709
                                710
                                711
                                712
                                713
                                714
                                715
                                716
                                717
                                718
                                719
                                720
                                721
                                722
                                723
                                724
                                725
                                726
                                727
                                728
                                729
                                730
                                731
                                732
                                733
                                734
                                735
                                736
                                737
                                738
                                739
                                740
                                741
                                742
                                743
                                744
                                745
                                746
                                747
                                748
                                749
                                750
                                751
                                752
                                753
                                754
                                755
                                756
                                757
                                758
                                759
                                760
                                761
                                762
                                763
                                764
                                765
                                766
                                767
                                768
                                769
                                770
                                771
                                772
                                773
                                774
                                775
                                776
                                777
                                778
                                779
                                780
                                781
                                782
                                783
                                784
                                785
                                786
                                787
                                788
                                789
                                790
                                791
                                792
                                793
                                794
                                795
                                796
                                797
                                798
                                799
                                800
                                801
                                802
                                803
                                804
                                805
                                806
                                807
                                808
                                809
                                810
                                811
                                812
                                813
                                814
                                815
                                816
                                817
                                818
                                819
                                820
                                821
                                822
                                823
                                824
                                825
                                826
                                827
                                828
                                829
                                830
                                831
                                832
                                833
                                834
                                835
                                836
                                837
                                838
                                839
                                840
                                841
                                842
                                843
                                844
                                845
                                846
                                847
                                848
                                849
                                850
                                851
                                852
                                853
                                854
                                855
                                856
                                857
                                858
                                859
                                860
                                861
                                862
                                863
                                864
                                865
                                866
                                867
                                868
                                869
                                870
                                871
                                872
                                873
                                874
                                875
                                876
                                877
                                878
                                879
                                880
                                881
                                882
                                883
                                884
                                885
                                886
                                887
                                888
                                889
                                890
                                891
                                892
                                893
                                894
                                895
                                896
                                897
                                898
                                899
                                900
                                901
                                902
                                903
                                904
                                905
                                906
                                907
                                908
                                909
                                910
                                911
                                912
                                913
                                914
                                915
                                916
                                917
                                918
                                919
                                920
                                921
                                922
                                923
                                924
                                925
                                926
                                927
                                928
                                929
                                930
                                931
                                932
                                933
                                934
                                935
                                936
                                937
                                938
                                939
                                940
                                941
                                942
                                943
                                944
                                945
                                946
                                947
                                948
                                949
                                950
                                951
                                952
                                953
                                954
                                955
                                956
                                957
                                958
                                959
                                960
                                961
                                962
                                963
                                964
                                965
                                966
                                967
                                968
                                969
                                970
                                971
                                972
                                973
                                974
                                975
                                976
                                977
                                978
                                979
                                980
                                981
                                982
                                983
                                984
                                985
                                986
                                987
                                988
                                989
                                990
                                991
                                992
                                993
                                994
                                995
                                996
                                997
                                998
                                999

```

530486-01

270189-62

ELUCON CHITTOJ ATAO 0.01

270189-63

02/18/86 PAGE 10

MCS-96 MACRO ASSEMBLER A2D_BUFFERING UTILITY

```

ERR LOC OBJECT LINE SOURCE STATEMENT
00AC 332 CSEG
00AC 333
00AC F2 334 A2D_DONE Vector: ; A/D INTERRUPT ROUTINE
00AC 335 PUSHF
00AC 336
00AD C60600 337 STB ad_result_lo,[top_of_buffer]+
00B0 C60600 338 STB ad_result_hi,[top_of_buffer]+
00B3 51070900 339 ANDI ad_command,Control_A2D,$000000111b ;load channel number
340
00B7 E00809 341 DJNZ sample_count, Sample_Again
00BA 1708 342 INCB sample_count
343
00BC 880406 344 CMP top_of_buffer,aductemp1 ; Check top of buffer
00BF DF26 345 BE Top_of_buffers
00C1 F3 346 POPF
00C2 F0 347 RET
348
00C3 349 Sample_Again:
00C3 640A02 350 ADD aductemp0,Sample_Period ; Set next sample time
00C6 880406 351 CMP top_of_buffer,aductemp1 ; Check top of buffer
352 ; for later jump
353
00CF 30080B 354 JBC Load_HSO_Command_Start_A2D
355 sample_count,0,Make_HSO_High
356
00D2 357 Make_HSO_low:
00D2 FD 358 nop ; wait 8 states after HSO load
359 Load_HSO_Command_HSO_0_Low
360 ; Load for change of HSO to trigger S/H
361 BE Top_of_buffers
362 POPF
363 RET
364
00D0 365 Make_HSO_high:
00D0 366 Load_HSO_Command_HSO_0_High ; Load for change of HSO to trigger S/H
367 BE Top_of_buffers
368 POPF
369 RET
370
00E3 DF02 371 Top_of_buffers:
00E3 F3 372 ANDI Control_A2D,$NOT(Busy) ; Clear converter BUSY bit
00E6 F0 373 POPF
00E6 374 RET
375
00E7 717F09 376 END
00EA F3 377
00EB F0 378
00EC 379
00EC 380
00EC 381
00EC 382
00EC 383
00EC 384
00EC 385

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

The listing contains a fairly complete description of what the program does. The block by block operations are shown below:

Lines 1-198 describe the program, declare the variables and set up equates. Several of these variables are declared as overlayable, so the user needs to be careful if using this module for other than the FFT program.

Lines 205-210 declare a macro which is used to load the HSO unit. This will be used repeatedly through the code.

Lines 212-253 determine whether a conversion or download has been requested. If a download has been requested, the data is downloaded to the destination array as either paired or linear data. Paired data has been described earlier.

Lines 255-278 contain a subroutine which converts the destination array to either signed or unsigned numbers. The numbers are also shifted right to provide the desired full-scale value as requested by SHIFT__COUNT.

Lines 279-334 initialize the conversion routine. HSO.0 is toggled with the start of each routine so that an external sample and hold can be used. The instructions in lines 308, 316, and 326 have been interweaved with the Load_HSO__Commands to provide the required 8 state delays between HSO loadings. If this was not done, NOPs would have been needed. It is easier to understand the code if these lines are thought of as being gathered at line 326.

Lines 337-353 are the actual A/D interrupt routine. The A/D results are placed BYTE by BYTE on the buffer, the A/D is reloaded, and then the number of samples taken is compared to the number needed. Note that the A/D command register needs to be reloaded even if the channel does not change. INCB on line 348 is used to insure that the DJNZ falls through on the next pass (if sample_count is not reset).

Lines 355-396 complete the routine. The HSO is set up to trigger the next conversion and provide the HSO.0 toggle for an external sample and hold. Once again, the time between consecutive loads of the HSO is 8 states minimum. Note that this section of code has been optimized for speed by reducing branches to an absolute minimum and duplicating code where needed.

This concludes the description of the A to D buffer module. In the FFT program, this module is run, then the FFT transform module, then the plot module. This allows variables to be overlaid, saving RAM space. The time cost for this is not bad, considering the printer is the limiting factor in these conversions. If more RAM

was provided, and the FFT was run with its data in external RAM, this module could be run simultaneously with the other modules.

10.0 DATA PLOTTING MODULE

The plot module is relatively straight-forward, and is shown in Listing 5. After the declarations, which include overlayable registers, an initialization routine is listed. This separately called routine sets up the serial port on the 8096 to talk to the printer. In this case, the port has to be set for 300 baud.

A console out routine follows. This routine can also be called by any program, but it is used only by the plot routine in this example. The write to port 1 is used to trace the program flow. The character to be output is passed to this routine on the stack. This conforms to PLM-96 requirements.

Since all stack operations on the 8096 are 16-bits wide, a multiple character feature has been added to the console out routine. If the high byte it receives is non-zero, the ASCII character in that byte is printed after the character in the low byte. If the high byte has a value between 128 and 255, the character in the low byte is repeated the number of times indicated by the least significant 7 bits of the high byte.

The print decimal number routine is next. It is called with two words on the stack. The first word is the unsigned value to be printed. The second byte contains information on the number of places to be printed and zero and blank suppression. This routine is not overflow-proof. The user must declare a sufficient number of places to be printed for all possible numbers.

The DRAW_GRAPH routine provides the plot. It first sends a series of carriage return, line feeds (CRLFs) to clear the printer and provides a margin on the paper. Each row is started with the row number, 2 spaces, and a "+". Asterisks are then plotted until

Number of asterisks > FFT Value / PLOT__RES

Recall that PLOT__RES is a variable set by the main program. When the number of asterisks hits the desired value, the value of the line is printed. If the Decibel mode is selected, the line value is divided by 512 and printed in integer + decimal part form, followed by "dB". If the number of asterisks reaches PLOT__MAX, no value is printed. The next line is then started. A line with only a "I" is printed before the next plot line to provide a more aesthetic display on the printer. If a CRT was used, this extra line would probably not be wanted.

MCS-96 MACRO ASSEMBLER PLOT_SERIAL

02/18/86

PAGE 1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:PLOTSP.A96

OBJECT FILE: :F2:PLOTSP.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```

ERR LOC OBJECT          LINE      SOURCE STATEMENT
1          $pagelength(50)
2
3      PLOT_SERIAL MODULE STACKSIZE (6)
4
5      ; Intel Corporation, December 12, 1985
6      ; by Ira Horden, MCO Applications
7
8      ; This program produces a plot on serially connected printer. The
9      ; magnitude of each of the 32 input values is plotted horizontally, with one
10     ; ":" followed by a linefeed between each plot line. Each plot line starts
11     ; with a "+" and the entire plot begins with 3 line feeds and ends with a form
12     ; feed. The values to be plotted are 32 unsigned words based at the externally
13     ; defined pointer PLOT_IN.
14
15     ; The routine INIT_OUTPUT must be run to set up the serial port when the
16     ; system is turned on. CON_OUT can be used by a program to output to the
17     ; serial port. DRAW_GRAPH is the routine that automatically plots the data.
18
19     ; Sizing of the graph can be done using PLOT_RES, which determines how many
20     ; units are needed for each dot, and PLOT_MAX, which is the maximum value the
21     ; program will be passed. Note that (PLOT_MAX/PLOT_RES) defines the maximum
22     ; number of columns the routine will print.
23
24
25     RSEG
26     EXTRN  iocl, baud_reg, spcon, spstat, sbuf, portl
27     EXTRN  zero, ax, bx, cx, dx, FFT_MODE
28     sptmp:  ddb 1
29
30     OSEG at 24H
31     value:   ddb 1
32     divisor: ddb 1
33     xptr:    ddb 1
34     yptr:    ddb 1
35     xval:    ddb 1
36     log_val: ddb 1
37
38     DSEG
39     EXTRN  PLOT_IN
40
41     $ject

```

270189-64

MCS-96 MACRO ASSEMBLER PLOT_SERIAL

02/18/86

PAGE 2

```

ERR LOC OBJECT      LINE      SOURCE STATEMENT
2500                42
43      CSEG at 2500H      ;;;      PROGRAM MODULE BEGINS
44
45      PUBLIC INIT_OUTPUT, CON_OUT, DRAW_GRAPH
46      EXTRN PLOT_RES, PLOT_RES_2, PLOT_MAX
47
48      INIT_OUTPUT:      ; INITIALIZE SERIAL PORT
49
2500 B12000          E 50      ldb      iocl,#00100000B ; set p2.0 to txd
51
0030               52      baud_val equ      624      ; 624=300 baud (at 12 MHz)
0270               53
0082               54      baud_high equ      ((baud_val-1)/256) OR 80H      ; set for XTALL clock
006F               55      baud_low equ      (baud_val-1) MOD 256
56
2503 B16F00          E 57      ldb      baud_reg,#baud_low
2506 B18200          E 58      ldb      baud_reg,#baud_high
59
2509 B14900          E 60      ldb      spcon,#01001001b      ; enable receiver mode 1
250C B12000          R 61      ldb      sptmp,#00100000B      ; set TI-tmp
62
250F F0              63      RET
64
65      $ject

```

270189-65

MCS-96 MACRO ASSEMBLER PLOT_SERIAL

02/18/86

PAGE 3

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			66	
			67	
			68	CONSOLE OUT ROUTINE
			69	
			70	Call with a word parameter on stack. The low byte has the character
			71	to be sent. If the high byte has a value between 81H and 8FEH, the
			72	character is repeated 1 to 126 times respectively. One repeat means
			73	that the character will be printed 2 times. If the high byte contains
			74	a value between 1 and 7FH, the character represented by that value will
			75	be printed after the character in the low byte. If the high byte
			76	contains a value of zero only the low byte will be printed.
			77	
	2510		78	CON_OUT:
	2510	CC00	79	pop ax ; cx contains the calling address
	2512	CC00	80	pop dx ;
	2514	3F011C	81	jbs dx+1,7,onechr ; If bit 7 is set print one character
	2517	980001	82	cmph dx+1,zero
	251A	DF17	83	je onechr ; if highbyte=0 print one character
			84	
	251C	900000	85	twochr: orb sptmp,sptat ; wait for TI
	251F	3500FA	86	jbc sptmp,5,twochr
	2522	71DF00	87	andb sptmp,#11011111b ; clear TI-tmp
	2525	900000	88	orb zero,sptat ; remove possible false TI
			89	
	2528	B00000	90	ldb sbuf,dx
	252B	B00100	91	ldb dx,dx+1 ; Load second character
	252E	1101	92	clrb dx+1 ; clear count byte
	2530	717F00	93	andb dx,#07FH ; mask MSB
			94	
	2533	1701	95	onechr: inch dx+1
	2535	717F01	96	andb dx+1,#7FH
	2538	900000	97	waitl: orb sptmp,sptat ; wait for TI
	253B	3500FA	98	jbc sptmp,5,waitl
	253E	71DF00	99	andb sptmp,#11011111b ; clear TI-tmp
	2541	900000	100	orb zero,sptat ; remove possible false TI
			101	
	2544	B00000	102	ldb sbuf,dx
	2547	E001EE	103	DJNZ dx+1,waitl
	254A	E300	104	BR [ax] ; Effectively a RET
			105	
			106	\$eject
			107	
			108	
			109	
			110	
			111	
			112	
			113	
			114	
			115	
			116	
			117	
			118	
			119	
			120	
			121	
			122	
			123	
			124	
			125	
			126	
			127	
			128	
			129	
			130	
			131	
			132	
			133	
			134	
			135	
			136	
			137	
			138	
			139	
			140	
			141	
			142	
			143	
			144	
			145	
			146	
			147	
			148	
			149	
			150	
			151	
			152	
			153	
			154	
			155	
			156	
			157	
			158	
			159	
			160	
			161	
			162	
			163	
			164	
			165	
			166	
			167	
			168	
			169	
			170	
			171	
			172	
			173	
			174	
			175	
			176	
			177	
			178	
			179	
			180	
			181	
			182	
			183	
			184	
			185	
			186	
			187	
			188	
			189	
			190	
			191	
			192	
			193	
			194	
			195	
			196	
			197	
			198	
			199	
			200	
			201	
			202	
			203	
			204	
			205	
			206	
			207	
			208	
			209	
			210	
			211	
			212	
			213	
			214	
			215	
			216	
			217	
			218	
			219	
			220	
			221	
			222	
			223	
			224	
			225	
			226	
			227	
			228	
			229	
			230	
			231	
			232	
			233	
			234	
			235	
			236	
			237	
			238	
			239	
			240	
			241	
			242	
			243	
			244	
			245	
			246	
			247	
			248	
			249	
			250	
			251	
			252	
			253	
			254	
			255	

270189-66

MCS-96 MACRO ASSEMBLER PLOT_SERIAL

02/18/86

PAGE 4

```

ERR LOC OBJECT          LINE      SOURCE STATEMENT
                                107 ;
                                108 ;
                                109 ;
                                110 ;      Call with two words on stack.  The first is the value to be printed.
                                111 ;      The second has mode information in the low byte.
                                112 ;      MODE:    000 = suppress all zeros
                                113 ;              001 = print all numbers
                                114 ;              010 = suppress all zeros except rightmost
                                115 ;              1xx = do not print leading blanks
                                116 ;
                                117 ;      The high byte of the 2nd word = 2x the number of places to be printed
                                118 ;
                                119 ;
                                120 PRINT_NUM:      ; Send Decimal number to CON_OUT
                                121     pop         cx      ; cx = # of digits
                                122     pop         bx      ; bx is mode byte, bx+1 is divisor pointer
                                123     ldbw     dx,bx+1
                                124     ld         divisor,divtab[dx]
                                125     pop         value
                                126     div_loop:
                                127     clr         value+2
                                128     divu     value,divisor      ; divide ax,dx by divisor
                                129     jbs     bx,0,chr_ok      ; print character regardless of value
                                130     cmpb     value,zero
                                131     jne     non_0      ; jump if value is non zero
                                132     Val_0:      ; Value is zero
                                133     jbc     bx,1,prntsp      ; Print space instead of 0
                                134     jbs     divisor,0,chr_ok      ; If in rightmost position print 0
                                135     prntsp:  jbs     bx,2,cont      ; Do not print space if bit is set
                                136     ld         value,$0F0H      ; 0F0h+30h = 20H = space
                                137     br         chr_ok
                                138
                                139     non_0:  orb     bx,$0001B      ; Set flag so 0's will be printed
                                140     chr_ok:  add     value,$30h      ; 30h + n = 0 to 9 ascii
                                141     and     value,$7Fh      ; send least sig seven bits, clear upper word
                                142     CON_OUT:  push    value
                                143     call     con_out      ; output ascii result (result<9)
                                144     cont:   ld     value,value+2      ; load value with remainder
                                145     clr     divisor+2
                                146     divu     divisor,$10      ; next lower power of ten
                                147     cmp     divisor,zero
                                148     jne     div_loop
                                149     div_done:
                                150     br     [cx]
                                151
                                152     DIVTAB:      ;      Number of places for result
                                153     dcw     0, 1, 10, 100, 1000, 10000      ; divisor table - 10**n

```

270189-67

MCS-96 MACRO ASSEMBLER PLOT_SERIAL

02/18/86

PAGE 5

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			154	
			155	
			156	
			157	
	25A2		158	DRAW_GRAPH: ; Graph drawing routine
	25A2 C90D00		159	push #0dh
	25A5 2F69		160	call con_out
	25A7 C90A82		161	push #820AH ; Clear 3 lines
	25AA 2F64		162	call CON_OUT
	25AC C90000		163	push #00
	25AF 2F5F		164	call CON_out
			165	
	25B1 012C		166	clr xptr
	25B3 0130		167	clr xval
	25B5		168	NXT_ROW: ;
	25B5 C90D0A		169	push #0A0DH ; CRLF
	25B8 2F66		170	call CON_OUT
	25BA C90000		171	push #00H ; nul
	25BD 2F61		172	call CON_OUT
			173	
	25BF C830		174	push xval
	25C1 C9020A		175	push #0A00H or 0010b ; suppress all zeros except rightmost
	25C4 2F86		176	call PRINT_NUM
			177	
	25C6 C92020		178	push #2020H ; Print 2 spaces
	25C9 2F45		179	call CON_OUT
	25CB C92B00		180	push #2BH ;
	25CE 2F40		181	call con_out
			182	
	25D0 A100002E	E	183	ld yptr,\$PLOT_RES_2 ; PLOT_RES_2 = PLOT_RES/2
			184	
			185	; PLOT_RES is defined 7 lines down
	25D4		186	NXT_COL: ; Next Column
	25D4 8B2D00002E	E	187	cmp yptr,PLOT_IN[xptr]
	25D9 D911		188	jh PRT_NUM
	25DB		189	PRT_MK: ; Print Mark
	25DB C92A00		190	push #2AH
	25DE 2F30		191	call CON_OUT
	25E0		192	INC_CNT: ;
	25E0 6500002E	E	193	add yptr,\$PLOT_RES ; PLOT_RES = number of inputs per output point
	25E4 8900002E	E	194	cmp yptr,\$PLOT_MAX ; PLOT_max = maximum line length
	25E8 D1EA		195	jnh nxt_col
	25EA 204F		196	br NXTLN
			197	\$eject

270189-68

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			198	
		25EC	199	PRT_NUM:
		25EC 8900002E	200	cmp yptr,#PLOT_RES_2 ; If value is less then minimum needed
		25F0 DF49	201	be NXTLN ; for a plot, do not print value
			202	
		25F2 C92020	203	push #2020H ; print 2 spaces then value
		25F5 2F19	204	call con_out
		25F7 3B000B	205	JBS FFT_MODE,3,db_mode
			206	
		25FA	207	norm_mode:
		25FA CB2D0000	208	push PLOT_IN[xptr]
		25FE C9000A	209	push #(0A00H or 0000B) ; supress all zeros
		2601 2F49	210	call PRINT_NUM
		2603 2036	211	BR NXTLN
			212	
		2605	213	db_mode:
		2605 A32D00002E	214	ld yptr,plot_in[xptr] ; PLOT_IN = 512*10*LOG(x)
		260A 08012E	215	shr yptr,#1 ; yptr=265 * 10LOG(x)
		260D AC2F00	216	ldbze ax,yptr+1 ; ax= 10LOG(x) = yptr/256
			217	
		2610 C800	218	push ax ; Print AX
		2612 C9020A	219	push #(0A00H or 0010B) ; supress all but rightmost zero
		2615 2F35	220	call PRINT_NUM
		2617 C92E00	221	push #2EH ; Decimal point
		261A 2EF4	222	call con_out
			223	
		261C B02E01	224	ldb ax+1,yptr ; high byte of ax = fractional portion of
		261F 1100	225	clrb ax ; 10LOG(x)
			226	
		2621 6DE60300	227	mula ax,#3E6H ; if ax=FF00H then ax+2 now = 998 decimal
		2625 370102	228	jbc ax+1,7,no_rnd
		2628 0700	229	inc dx ; round value up
			230	
		262A C800	231	no_rnd: push dx ; dx=ax+2
		262C C90106	232	push #(600H or 0001B) ; print all numbers to three places
		262F 2F1B	233	call Print_num
		2631 C92000	234	push #20H ; space
		2634 2EDA	235	call con_out
		2636 C96442	236	push #4264H ; "dB"
		2639 2ED5	237	call con_out
			238	
		263A C90000	239	\$eject

270189-69

PAGE 7

270189-70

At the end of the plot, a form feed is given to set the printer up for the next graph. Our printer would frequently miss the character after a CRLF. To solve this problem, a null (ASCII 0) is sent after every CRLF to make sure the printer is ready for the next line. This has been found to be a problem with many devices running at close to their maximum capacity, and the nulls work well to solve it.

With the plot completed, the program begins to run again by taking another set of A to D samples.

11.0 USING THE FFT PROGRAM

The program can be used with either real or tabled data. If real data is used, the signal is applied to analog channel 1. The program as written performs A/D samples at 100 microsecond intervals, collecting the 64 samples in 6.4 milliseconds. This sets the sampling window frequency at 156 Hz. If tabled data is used, 64 words of data should be placed in the location pointed to by DATA0 in the TABLE_LOAD routine of the Main Module.

Program control is specified by FFT_MODE which is loaded in the main module. Also within the main module are settings which control the A to D buffer routine and the Plot routine. The intention was to have only one module to change and recompile to vary parameters in the entire program.

The program modules are set up to run one-at-a-time so that the code would be easy to understand. Additionally, the Plot routine takes so long relative to the other sections, that it doesn't pay to try to overlap code sections. If this code were to be converted to run a process instead of print a graph, it might be worthwhile to run the FFT and the A/D routines at the same time.

If the goal of a modified program is to have the highest frequency sampling possible, it might be desirable to streamline the A/D section and run it without interruption. When the A to D routine was complete the FFT routine could be started. The reasoning behind this is that at the fastest A/D speeds the processor will be almost completely tied up processing the A/D information and storing it away. Using an interrupt based A/D routine would slow things down.

A set of programs which will perform a FFT has been presented in this application note. These programs are available from the INSITE users library as program CA-26. More importantly, dozens of programing examples have been made available, making it easier to get started with the 8096. Examples of how to use the hardware on the 8096 have already appeared in AP-248, "Using The 8096". These two applications notes form a good base for the understanding of MCS-96 microcontroller based design.

Listing 8—The Plot Module (Continued)

```

      1000  LD  A,0
      1010  LD  B,0
      1020  LD  C,0
      1030  LD  D,0
      1040  LD  E,0
      1050  LD  F,0
      1060  LD  H,0
      1070  LD  L,0
      1080  LD  M,0
      1090  LD  N,0
      1100  LD  O,0
      1110  LD  P,0
      1120  LD  Q,0
      1130  LD  R,0
      1140  LD  S,0
      1150  LD  T,0
      1160  LD  U,0
      1170  LD  V,0
      1180  LD  W,0
      1190  LD  X,0
      1200  LD  Y,0
      1210  LD  Z,0
      1220  LD  AA,0
      1230  LD  AB,0
      1240  LD  AC,0
      1250  LD  AD,0
      1260  LD  AE,0
      1270  LD  AF,0
      1280  LD  AG,0
      1290  LD  AH,0
      1300  LD  AI,0
      1310  LD  AJ,0
      1320  LD  AK,0
      1330  LD  AL,0
      1340  LD  AM,0
      1350  LD  AN,0
      1360  LD  AO,0
      1370  LD  AP,0
      1380  LD  AQ,0
      1390  LD  AR,0
      1400  LD  AS,0
      1410  LD  AT,0
      1420  LD  AU,0
      1430  LD  AV,0
      1440  LD  AW,0
      1450  LD  AX,0
      1460  LD  AY,0
      1470  LD  AZ,0
      1480  LD  BA,0
      1490  LD  BB,0
      1500  LD  BC,0
      1510  LD  BD,0
      1520  LD  BE,0
      1530  LD  BF,0
      1540  LD  BG,0
      1550  LD  BH,0
      1560  LD  BI,0
      1570  LD  BJ,0
      1580  LD  BK,0
      1590  LD  BL,0
      1600  LD  BM,0
      1610  LD  BN,0
      1620  LD  BO,0
      1630  LD  BP,0
      1640  LD  BQ,0
      1650  LD  BR,0
      1660  LD  BS,0
      1670  LD  BT,0
      1680  LD  BU,0
      1690  LD  BV,0
      1700  LD  BW,0
      1710  LD  BX,0
      1720  LD  BY,0
      1730  LD  BZ,0
      1740  LD  CA,0
      1750  LD  CB,0
      1760  LD  CC,0
      1770  LD  CD,0
      1780  LD  CE,0
      1790  LD  CF,0
      1800  LD  CG,0
      1810  LD  CH,0
      1820  LD  CI,0
      1830  LD  CJ,0
      1840  LD  CK,0
      1850  LD  CL,0
      1860  LD  CM,0
      1870  LD  CN,0
      1880  LD  CO,0
      1890  LD  CP,0
      1900  LD  CQ,0
      1910  LD  CR,0
      1920  LD  CS,0
      1930  LD  CT,0
      1940  LD  CU,0
      1950  LD  CV,0
      1960  LD  CW,0
      1970  LD  CX,0
      1980  LD  CY,0
      1990  LD  CZ,0
      2000  LD  DA,0
      2010  LD  DB,0
      2020  LD  DC,0
      2030  LD  DD,0
      2040  LD  DE,0
      2050  LD  DF,0
      2060  LD  DG,0
      2070  LD  DH,0
      2080  LD  DI,0
      2090  LD  DJ,0
      2100  LD  DK,0
      2110  LD  DL,0
      2120  LD  DM,0
      2130  LD  DN,0
      2140  LD  DO,0
      2150  LD  DP,0
      2160  LD  DQ,0
      2170  LD  DR,0
      2180  LD  DS,0
      2190  LD  DT,0
      2200  LD  DU,0
      2210  LD  DV,0
      2220  LD  DW,0
      2230  LD  DX,0
      2240  LD  DY,0
      2250  LD  DZ,0
      2260  LD  EA,0
      2270  LD  EB,0
      2280  LD  EC,0
      2290  LD  ED,0
      2300  LD  EE,0
      2310  LD  EF,0
      2320  LD  EG,0
      2330  LD  EH,0
      2340  LD  EI,0
      2350  LD  EJ,0
      2360  LD  EK,0
      2370  LD  EL,0
      2380  LD  EM,0
      2390  LD  EN,0
      2400  LD  EO,0
      2410  LD  EP,0
      2420  LD  EQ,0
      2430  LD  ER,0
      2440  LD  ES,0
      2450  LD  ET,0
      2460  LD  EU,0
      2470  LD  EV,0
      2480  LD  EW,0
      2490  LD  EX,0
      2500  LD  EY,0
      2510  LD  EZ,0
      2520  LD  FA,0
      2530  LD  FB,0
      2540  LD  FC,0
      2550  LD  FD,0
      2560  LD  FE,0
      2570  LD  FF,0
      2580  LD  FG,0
      2590  LD  FH,0
      2600  LD  FI,0
      2610  LD  FJ,0
      2620  LD  FK,0
      2630  LD  FL,0
      2640  LD  FM,0
      2650  LD  FN,0
      2660  LD  FO,0
      2670  LD  FP,0
      2680  LD  FQ,0
      2690  LD  FR,0
      2700  LD  FS,0
      2710  LD  FT,0
      2720  LD  FU,0
      2730  LD  FV,0
      2740  LD  FW,0
      2750  LD  FX,0
      2760  LD  FY,0
      2770  LD  FZ,0
      2780  LD  GA,0
      2790  LD  GB,0
      2800  LD  GC,0
      2810  LD  GD,0
      2820  LD  GE,0
      2830  LD  GF,0
      2840  LD  GG,0
      2850  LD  GH,0
      2860  LD  GI,0
      2870  LD  GJ,0
      2880  LD  GK,0
      2890  LD  GL,0
      2900  LD  GM,0
      2910  LD  GN,0
      2920  LD  GO,0
      2930  LD  GP,0
      2940  LD  GQ,0
      2950  LD  GR,0
      2960  LD  GS,0
      2970  LD  GT,0
      2980  LD  GU,0
      2990  LD  GV,0
      3000  LD  GW,0
      3010  LD  GX,0
      3020  LD  GY,0
      3030  LD  GZ,0
      3040  LD  HA,0
      3050  LD  HB,0
      3060  LD  HC,0
      3070  LD  HD,0
      3080  LD  HE,0
      3090  LD  HF,0
      3100  LD  HG,0
      3110  LD  HH,0
      3120  LD  HI,0
      3130  LD  HJ,0
      3140  LD  HK,0
      3150  LD  HL,0
      3160  LD  HM,0
      3170  LD  HN,0
      3180  LD  HO,0
      3190  LD  HP,0
      3200  LD  HQ,0
      3210  LD  HR,0
      3220  LD  HS,0
      3230  LD  HT,0
      3240  LD  HU,0
      3250  LD  HV,0
      3260  LD  HW,0
      3270  LD  HX,0
      3280  LD  HY,0
      3290  LD  HZ,0
      3300  LD  IA,0
      3310  LD  IB,0
      3320  LD  IC,0
      3330  LD  ID,0
      3340  LD  IE,0
      3350  LD  IF,0
      3360  LD  IG,0
      3370  LD  IH,0
      3380  LD  II,0
      3390  LD  IJ,0
      3400  LD  IK,0
      3410  LD  IL,0
      3420  LD  IM,0
      3430  LD  IN,0
      3440  LD  IO,0
      3450  LD  IP,0
      3460  LD  IQ,0
      3470  LD  IR,0
      3480  LD  IS,0
      3490  LD  IT,0
      3500  LD  IU,0
      3510  LD  IV,0
      3520  LD  IW,0
      3530  LD  IX,0
      3540  LD  IY,0
      3550  LD  IZ,0
      3560  LD  JA,0
      3570  LD  JB,0
      3580  LD  JC,0
      3590  LD  JD,0
      3600  LD  JE,0
      3610  LD  JF,0
      3620  LD  JG,0
      3630  LD  JH,0
      3640  LD  JI,0
      3650  LD  JJ,0
      3660  LD  JK,0
      3670  LD  JL,0
      3680  LD  JM,0
      3690  LD  JN,0
      3700  LD  JO,0
      3710  LD  JP,0
      3720  LD  JQ,0
      3730  LD  JR,0
      3740  LD  JS,0
      3750  LD  JT,0
      3760  LD  JU,0
      3770  LD  JV,0
      3780  LD  JW,0
      3790  LD  JX,0
      3800  LD  JY,0
      3810  LD  JZ,0
      3820  LD  KA,0
      3830  LD  KB,0
      3840  LD  KC,0
      3850  LD  KD,0
      3860  LD  KE,0
      3870  LD  KF,0
      3880  LD  KG,0
      3890  LD  KH,0
      3900  LD  KI,0
      3910  LD  KJ,0
      3920  LD  KK,0
      3930  LD  KL,0
      3940  LD  KM,0
      3950  LD  KN,0
      3960  LD  KO,0
      3970  LD  KP,0
      3980  LD  KQ,0
      3990  LD  KR,0
      4000  LD  KS,0
      4010  LD  KT,0
      4020  LD  KU,0
      4030  LD  KV,0
      4040  LD  KW,0
      4050  LD  KX,0
      4060  LD  KY,0
      4070  LD  KZ,0
      4080  LD  LA,0
      4090  LD  LB,0
      4100  LD  LC,0
      4110  LD  LD,0
      4120  LD  LE,0
      4130  LD  LF,0
      4140  LD  LG,0
      4150  LD  LH,0
      4160  LD  LI,0
      4170  LD  LJ,0
      4180  LD  LK,0
      4190  LD  LL,0
      4200  LD  LM,0
      4210  LD  LN,0
      4220  LD  LO,0
      4230  LD  LP,0
      4240  LD  LQ,0
      4250  LD  LR,0
      4260  LD  LS,0
      4270  LD  LT,0
      4280  LD  LU,0
      4290  LD  LV,0
      4300  LD  LW,0
      4310  LD  LX,0
      4320  LD  LY,0
      4330  LD  LZ,0
      4340  LD  MA,0
      4350  LD  MB,0
      4360  LD  MC,0
      4370  LD  MD,0
      4380  LD  ME,0
      4390  LD  MF,0
      4400  LD  MG,0
      4410  LD  MH,0
      4420  LD  MI,0
      4430  LD  MJ,0
      4440  LD  MK,0
      4450  LD  ML,0
      4460  LD  MM,0
      4470  LD  MN,0
      4480  LD  MO,0
      4490  LD  MP,0
      4500  LD  MQ,0
      4510  LD  MR,0
      4520  LD  MS,0
      4530  LD  MT,0
      4540  LD  MU,0
      4550  LD  MV,0
      4560  LD  MW,0
      4570  LD  MX,0
      4580  LD  MY,0
      4590  LD  MZ,0
      4600  LD  NA,0
      4610  LD  NB,0
      4620  LD  NC,0
      4630  LD  ND,0
      4640  LD  NE,0
      4650  LD  NF,0
      4660  LD  NG,0
      4670  LD  NH,0
      4680  LD  NI,0
      4690  LD  NJ,0
      4700  LD  NK,0
      4710  LD  NL,0
      4720  LD  NM,0
      4730  LD  NN,0
      4740  LD  NO,0
      4750  LD  NP,0
      4760  LD  NQ,0
      4770  LD  NR,0
      4780  LD  NS,0
      4790  LD  NT,0
      4800  LD  NU,0
      4810  LD  NV,0
      4820  LD  NW,0
      4830  LD  NX,0
      4840  LD  NY,0
      4850  LD  NZ,0
      4860  LD  OA,0
      4870  LD  OB,0
      4880  LD  OC,0
      4890  LD  OD,0
      4900  LD  OE,0
      4910  LD  OF,0
      4920  LD  OG,0
      4930  LD  OH,0
      4940  LD  OI,0
      4950  LD  OJ,0
      4960  LD  OK,0
      4970  LD  OL,0
      4980  LD  OM,0
      4990  LD  ON,0
      5000  LD  OO,0
      5010  LD  OP,0
      5020  LD  OQ,0
      5030  LD  OR,0
      5040  LD  OS,0
      5050  LD  OT,0
      5060  LD  OU,0
      5070  LD  OV,0
      5080  LD  OW,0
      5090  LD  OX,0
      5100  LD  OY,0
      5110  LD  OZ,0
      5120  LD  PA,0
      5130  LD  PB,0
      5140  LD  PC,0
      5150  LD  PD,0
      5160  LD  PE,0
      5170  LD  PF,0
      5180  LD  PG,0
      5190  LD  PH,0
      5200  LD  PI,0
      5210  LD  PJ,0
      5220  LD  PK,0
      5230  LD  PL,0
      5240  LD  PM,0
      5250  LD  PN,0
      5260  LD  PO,0
      5270  LD  PP,0
      5280  LD  PQ,0
      5290  LD  PR,0
      5300  LD  PS,0
      5310  LD  PT,0
      5320  LD  PU,0
      5330  LD  PV,0
      5340  LD  PW,0
      5350  LD  PX,0
      5360  LD  PY,0
      5370  LD  PZ,0
      5380  LD  QA,0
      5390  LD  QB,0
      5400  LD  QC,0
      5410  LD  QD,0
      5420  LD  QE,0
      5430  LD  QF,0
      5440  LD  QG,0
      5450  LD  QH,0
      5460  LD  QI,0
      5470  LD  QJ,0
      5480  LD  QK,0
      5490  LD  QL,0
      5500  LD  QM,0
      5510  LD  QN,0
      5520  LD  QO,0
      5530  LD  QP,0
      5540  LD  QQ,0
      5550  LD  QR,0
      5560  LD  QS,0
      5570  LD  QT,0
      5580  LD  QU,0
      5590  LD  QV,0
      5600  LD  QW,0
      5610  LD  QX,0
      5620  LD  QY,0
      5630  LD  QZ,0
      5640  LD  RA,0
      5650  LD  RB,0
      5660  LD  RC,0
      5670  LD  RD,0
      5680  LD  RE,0
      5690  LD  RF,0
      5700  LD  RG,0
      5710  LD  RH,0
      5720  LD  RI,0
      5730  LD  RJ,0
      5740  LD  RK,0
      5750  LD  RL,0
      5760  LD  RM,0
      5770  LD  RN,0
      5780  LD  RO,0
      5790  LD  RP,0
      5800  LD  RQ,0
      5810  LD  RR,0
      5820  LD  RS,0
      5830  LD  RT,0
      5840  LD  RU,0
      5850  LD  RV,0
      5860  LD  RW,0
      5870  LD  RX,0
      5880  LD  RY,0
      5890  LD  RZ,0
      5900  LD  SA,0
      5910  LD  SB,0
      5920  LD  SC,0
      5930  LD  SD,0
      5940  LD  SE,0
      5950  LD  SF,0
      5960  LD  SG,0
      5970  LD  SH,0
      5980  LD  SI,0
      5990  LD  SJ,0
      6000  LD  SK,0
      6010  LD  SL,0
      6020  LD  SM,0
      6030  LD  SN,0
      6040  LD  SO,0
      6050  LD  SP,0
      6060  LD  SQ,0
      6070  LD  SR,0
      6080  LD  SS,0
      6090  LD  ST,0
      6100  LD  SU,0
      6110  LD  SV,0
      6120  LD  SW,0
      6130  LD  SX,0
      6140  LD  SY,0
      6150  LD  SZ,0
      6160  LD  TA,0
      6170  LD  TB,0
      6180  LD  TC,0
      6190  LD  TD,0
      6200  LD  TE,0
      6210  LD  TF,0
      6220  LD  TG,0
      6230  LD  TH,0
      6240  LD  TI,0
      6250  LD  TJ,0
      6260  LD  TK,0
      6270  LD  TL,0
      6280  LD  TM,0
      6290  LD  TN,0
      6300  LD  TO,0
      6310  LD  TP,0
      6320  LD  TQ,0
      6330  LD  TR,0
      6340  LD  TS,0
      6350  LD  TT,0
      6360  LD  TU,0
      6370  LD  TV,0
      6380  LD  TW,0
      6390  LD  TX,0
      6400  LD  TY,0
      6410  LD  TZ,0
      6420  LD  UA,0
      6430  LD  UB,0
      6440  LD  UC,0
      6450  LD  UD,0
      6460  LD  UE,0
      6470  LD  UF,0
      6480  LD  UG,0
      6490  LD  UH,0
      6500  LD  UI,0
      6510  LD  UJ,0
      6520  LD  UK,0
      6530  LD  UL,0
      6540  LD  UM,0
      6550  LD  UN,0
      6560  LD  UO,0
      6570  LD  UP,0
      6580  LD  UQ,0
      6590  LD  UR,0
      6600  LD  US,0
      6610  LD  UT,0
      6620  LD  UU,0
      6630  LD  UV,0
      6640  LD  UW,0
      6650  LD  UX,0
      6660  LD  UY,0
      6670  LD  UZ,0
      6680  LD  VA,0
      6690  LD  VB,0
      6700  LD  VC,0
      6710  LD  VD,0
      6720  LD  VE,0
      6730  LD  VF,0
      6740  LD  VG,0
      6750  LD  VH,0
      6760  LD  VI,0
      6770  LD  VJ,0
      6780  LD  VK,0
      6790  LD  VL,0
      6800  LD  VM,0
      6810  LD  VN,0
      6820  LD  VO,0
      6830  LD  VP,0
      6840  LD  VQ,0
      6850  LD  VR,0
      6860  LD  VS,0
      6870  LD  VT,0
      6880  LD  VU,0
      6890  LD  VV,0
      6900  LD  VW,0
      6910  LD  VX,0
      6920  LD  VY,0
      6930  LD  VZ,0
      6940  LD  WA,0
      6950  LD  WB,0
      6960  LD  WC,0
      6970  LD  WD,0
      6980  LD  WE,0
      6990  LD  WF,0
      7000  LD  WG,0
      7010  LD  WH,0
      7020  LD  WI,0
      7030  LD  WJ,0
      7040  LD  WK,0
      7050  LD  WL,0
      7060  LD  WM,0
      7070  LD  WN,0
      7080  LD  WO,0
      7090  LD  WP,0
      7100  LD  WQ,0
      7110  LD  WR,0
      7120  LD  WS,0
      7130  LD  WT,0
      7140  LD  WU,0
      7150  LD  WV,0
      7160  LD  WW,0
      7170  LD  WX,0
      7180  LD  WY,0
      7190  LD  WZ,0
      7200  LD  XA,0
      7210  LD  XB,0
      7220  LD  XC,0
      7230  LD  XD,0
      7240  LD  XE,0
      7250  LD  XF,0
      7260  LD  XG,0
      7270  LD  XH,0
      7280  LD  XI,0
      7290  LD  XJ,0
      7300  LD  XK,0
      7310  LD  XL,0
      7320  LD  XM,0
      7330  LD  XN,0
      7340  LD  XO,0
      7350  LD  XP,0
      7360  LD  XQ,0
      7370  LD  XR,0
      7380  LD  XS,0
      7390  LD  XT,0
      7400  LD  XU,0
      7410  LD  XV,0
      7420  LD  XW,0
      7430  LD  XX,0
      7440  LD  XY,0
      7450  LD  XZ,0
      7460  LD  YA,0
      7470  LD  YB,0
      7480  LD  YC,0
      7490  LD  YD,0
      7500  LD  YE,0
      7510  LD  YF,0
      7520  LD  YG,0
      7530  LD  YH,0
      7540  LD  YI,0
      7550  LD  YJ,0
      7560  LD  YK,0
      7570  LD  YL,0
      7580  LD  YM,0
      7590  LD  YN,0
      7600  LD  YO,0
      7610  LD  YP,0
      7620  LD  YQ,0
      7630  LD  YR,0
      7640  LD  YS,0
      7650  LD  YT,0
      7660  LD  YU,0
      7670  LD  YV,0
      7680  LD  YW,0
      7690  LD  YX,0
      7700  LD  YY,0
      7710  LD  YZ,0
      7720  LD  ZA,0
      7730  LD  ZB,0
      7740  LD  ZC,0
      7750  LD  ZD,0
      7760  LD  ZE,0
      7770  LD  ZF,0
      7780  LD  ZG,0
      7790  LD  ZH,0
      7800  LD  ZI,0
      7810  LD  ZJ,0
      7820  LD  ZK,0
      7830  LD  ZL,0
      7840  LD  ZM,0
      7850  LD  ZN,0
      7860  LD  ZO,0
      7870  LD  ZP,0
      7880  LD  ZQ,0
      7890  LD  ZR,0
      7900  LD  ZS,0
      7910  LD  ZT,0
      7920  LD  ZU,0
      7930  LD  ZV,0
      7940  LD  ZW,0
      7950  LD  ZX,0
      7960  LD  ZY,0
      7970  LD  ZZ,0
      7980  LD  AA,0
      7990  LD  AB,0
      8000  LD  AC,0
      8010  LD  AD,0
      8020  LD  AE,0
      8030  LD  AF,0
      8040  LD  AG,0
      8050  LD  AH,0
      8060  LD  AI,0
      8070  LD  AJ,0
      8080  LD  AK,0
      8090  LD  AL,0
      8100  LD  AM,0
      8110  LD  AN,0
      8120  LD  AO,0
      8130  LD  AP,0
      8140  LD  AQ,0
      8150  LD  AR,0
      8160  LD  AS,0
      8170  LD  AT,0
      8180  LD  AU,0
      8190  LD  AV,0
      8200  LD  AW,0
      8210  LD  AX,0
      8220  LD  AY,0
      8230  LD  AZ,0
      8240  LD  BA,0
      8250  LD  BB,0
      8260  LD  BC,0
      8270  LD  BD,0
      8280  LD  BE,0
      8290  LD  BF,0
      8300  LD  BG,0
      8310  LD  BH,0
      8320  LD  BI,0
      8330  LD  BJ,0
      8340  LD  BK,0
      8350  LD  BL,0
      8360  LD  BM,0
      8370  LD  BN,0
      8380  LD  BO,0
      8390  LD  BP,0
      8400  LD  BQ,0
      8410  LD  BR,0
      8420  LD  BS,0
      8430  LD  BT,0
      8440  LD  BU,0
      8450  LD  BV,0
      8460  LD  BW,0
      8470  LD  BX,0
      8480  LD  BY,0
      8490  LD  BZ,0
      8500  LD  CA,0
      8510  LD  CB,0
      8520  LD  CC,0
      8530  LD  CD,0
      8540  LD  CE,0
      8550  LD  CF,0
      8560  LD  CG,0
      8570  LD  CH,0
      8580  LD  CI,0
      8590  LD  CJ,0
      8600  LD  CK,0
      8610  LD  CL,0
      8620  LD  CM,0
      8630  LD  CN,0
      8640  LD  CO,0
      8650  LD  CP,0
      8660  LD  CQ,0
      8670  LD  CR,0
      8680  LD  CS,0
      8690  LD  CT,0
      8700  LD  CU,0
      8710  LD  CV,0
      8720  LD  CW,0
      8730  LD  CX,0
      8740  LD  CY,0
      8750  LD  CZ,0
      8760  LD  DA,0
      8770  LD  DB,0
      8780  LD  DC,0
      8790  LD  DD,0
      8800  LD  DE,0
      8810  LD  DF,0
      8820  LD  DG,0
      8830  LD  DH,0
      8840  LD  DI,0
      8850  LD  DJ,0
      8860  LD  DK,0
      8870  LD  DL,0
      8880  LD  DM,0
      8890  LD  DN,0
      8900  LD  DO,0
      8910  LD  DP,0
      8920  LD  DQ,0
      8930  LD  DR,0
      8940  LD  DS,0
      8950  LD  DT,0
      8960  LD  DU,0
      8970  LD  DV,0
      8980  LD  DW,0
      8990  LD  DX,0
      9000  LD  DY,0
      9010  LD  DZ,0
      9020  LD  EA,0
      9030  LD  EB,0
      9040  LD  EC,0
      9050  LD  ED,0
      9060  LD  EE,0
      9070  LD  EF,0
      9080  LD  EG,0
      9090  LD  EH,0
      9100  LD  EI,0
      9110  LD  EJ,0
      9120  LD  EK,0
      9130  LD  EL,0
      9140  LD  EM,0
      9150  LD  EN,0
      9160  LD  EO,0
      9170  LD  EP,0
      9180  LD  EQ,0
      9190  LD  ER,0
      9200  LD  ES,0
      9210  LD  ET,0
      9220  LD  EU,0
      9230  LD  EV,0
      9240  LD  EW,0
      9250  LD  EX,0
      9260  LD  EY,0
      9270  LD  EZ,0
      9280  LD  FA,0
      9290  LD  FB,0
      9300  LD  FC,0
      9310  LD  FD,0
      9320  LD  FE,0
      9330  LD  FF,0
      9340  LD  FG,0
      9350  LD  FH,0
      9360  LD  FI,0
      9370  LD  FJ,0
      9380  LD  FK,0
      9390  LD  FL,0
      9400  LD  FM,0
      9410  LD  FN,0
      9420  LD  FO,0
      9430  LD  FP,0
      9440  LD  FQ,0
      9450  LD  FR,0
      9460  LD  FS,0
      9470  LD  FT,0
      9480  LD  FU,0
      9490  LD  FV,0
      9500  LD  FW,0
      9510  LD  FX,0
      9520  LD  FY,0
      9530  LD  FZ,0
      9540  LD  GA,0
      9550  LD  GB,0
      9560  LD  GC,0
      9570  LD  GD,0
      9580  LD  GE,0
      9590  LD  GF,0
      9600  LD  GG,0
      9610  LD  GH,0
      9620  LD  GI,0
      9630  LD  GJ,0
      96
```


12.0 APPENDIX A - MATRICES

Matrices are a convenient way to express groups of equations. Consider the complex discrete Fourier Transform in equation 9, with $N = 4$.

$$Y_n = \sum_{k=0}^3 X(k) W^{nk} \quad n = 0, 1, 2, 3$$

This can be expanded to

$$\begin{aligned} Y(0) &= X(0) W^0 + X(1) W^0 + X(2) W^0 + X(3) W^0 \\ Y(1) &= X(0) W^0 + X(1) W^1 + X(2) W^2 + X(3) W^3 \\ Y(2) &= X(0) W^0 + X(1) W^2 + X(2) W^4 + X(3) W^6 \\ Y(3) &= X(0) W^0 + X(1) W^3 + X(2) W^6 + X(3) W^9 \end{aligned}$$

In matrix notation, this is shown as

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

The first step to simplifying this is to reduce the center matrix. Recalling that

$$W^N = W^{N \bmod N} \quad \text{and} \quad W^0 = 1$$

The matrix can be reduced to have less non-trivial multiplications.

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

The square matrix can be factored into

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

For this equation to work, the $Y(1)$ and $Y(2)$ terms need to be swapped, as shown above. This procedure is a Bit Reversal, as described in the text.

Multiplying the two rightmost matrices results in

$$\begin{aligned} X(0) + X(2) W^0 \\ X(1) + X(3) W^0 \end{aligned} \quad \begin{array}{l} \text{requiring 4 complex multiplications} \\ \text{\& 4 complex additions} \end{array}$$

Noting that $W^0 = -W^2$, 2 of the complex multiplications can be eliminated, with the following results

$$\begin{aligned} X(0) + X(2) W^0 \\ X(1) + X(3) W^0 \end{aligned} \quad \begin{array}{l} \text{requiring 2 complex multiplications} \\ \text{and 4 complex additions} \end{array}$$

Since $W^1 = -W^3$, a similar result occurs when this vector is multiplied by the remaining square matrix. The resulting equations are:

$$\begin{aligned} Y(0) &= (X(0) + X(2) W^0) + W^0 (X(1) + X(3) W^0) \\ Y(2) &= (X(0) + X(2) W^0) - W^0 (X(1) + X(3) W^0) \\ Y(1) &= (X(0) - X(2) W^0) + W^1 (X(1) - X(3) W^0) \\ Y(3) &= (X(0) - X(2) W^0) - W^1 (X(1) - X(3) W^0) \end{aligned}$$

The number of complex multiplications required is 4, as compared with 16 for the unfactored matrix.

In general, the FFT requires

$$\frac{N * \text{EXPONENT}}{2} \text{ complex multiplications}$$

and

$$N * \text{EXPONENT} \text{ complex additions}$$

where

$$\text{EXPONENT} = \log_2 N$$

A standard Fourier Transform requires

$$N^2 \text{ complex multiplications}$$

and

$$N(N-1) \text{ complex additions}$$

13.0 APPENDIX B - PLOTS

The following plots are examples of output from the FFT program. These plots were generated using tabled data, but very similar plots have also been made using the analog input module. Typically, a plot made using the analog input module will not show quite as much power at each frequency and will show a positive value for the DC component. This is because it is difficult to get exactly a full-scale analog input with no DC offset.

Plot 1 is a Magnitude plot of a square wave of period NT.

Plot 2 is the same data plotted in dB. Note how the dB plot enhances the difference in the small signal values at the high frequencies.

Plot 3 shows the windowed version of this data. Note that the widening of the bins due to windowing shows energy in the even harmonics that is not actually present. For data of this type a different window other than Hanning would normally be used. Many window types are available, the selection of which can be determined by the type of data to be plotted.³

Plot 4 shows a sine wave of period NT/7 or frequency 7/NT.

Plot 5 shows the same input with windowing. Note the signal shown in bins 6 and 8.

Plot 6 shows a sine wave of period NT/7.5. Note the noise caused by the discontinuity as discussed earlier.

Plot 7 uses windowing on the data used for plot 6. Note the cleaner appearance.

Plot 8 shows a sine wave input of magnitude 0.707 and period NT/7.5.

Plot 9 shows same input with windowing.

Plot 10 shows a sine wave of magnitude 0.707/16 and period NT/11.

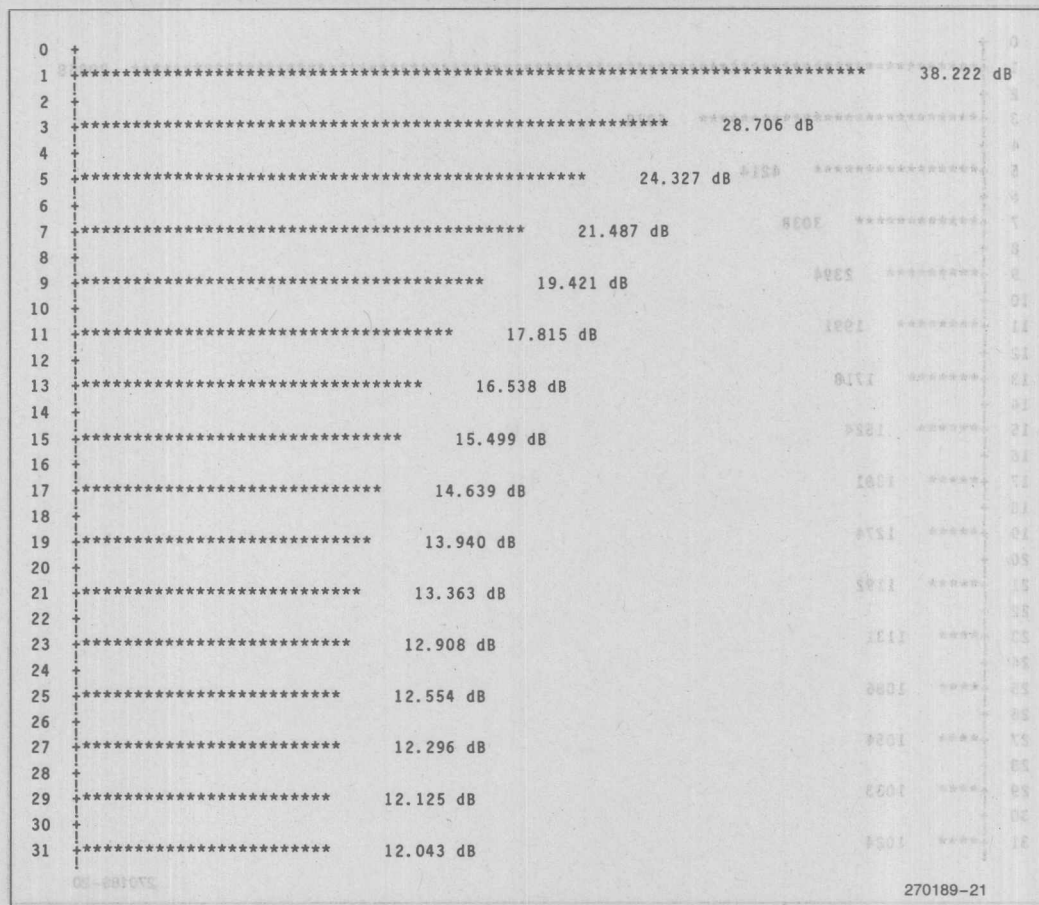
Plot 11 shows the same input with windowing. Note that there is no power shown in bins 10 and 12. This is because at 6 dB down from 3 dB they are nearly equal to zero.

Plot 12 uses the sum of the signals for plots 8 and 10 as inputs. Note that the component at period NT/11 is almost hidden.

Plot 13 uses the same signal as plot 12 but applies windowing. Now the period component at NT/11 can easily be seen. The Hanning window works well in this case to separate the signal from the leakage. If the signals were closer together the Hanning window may not have worked and another window may have been needed.

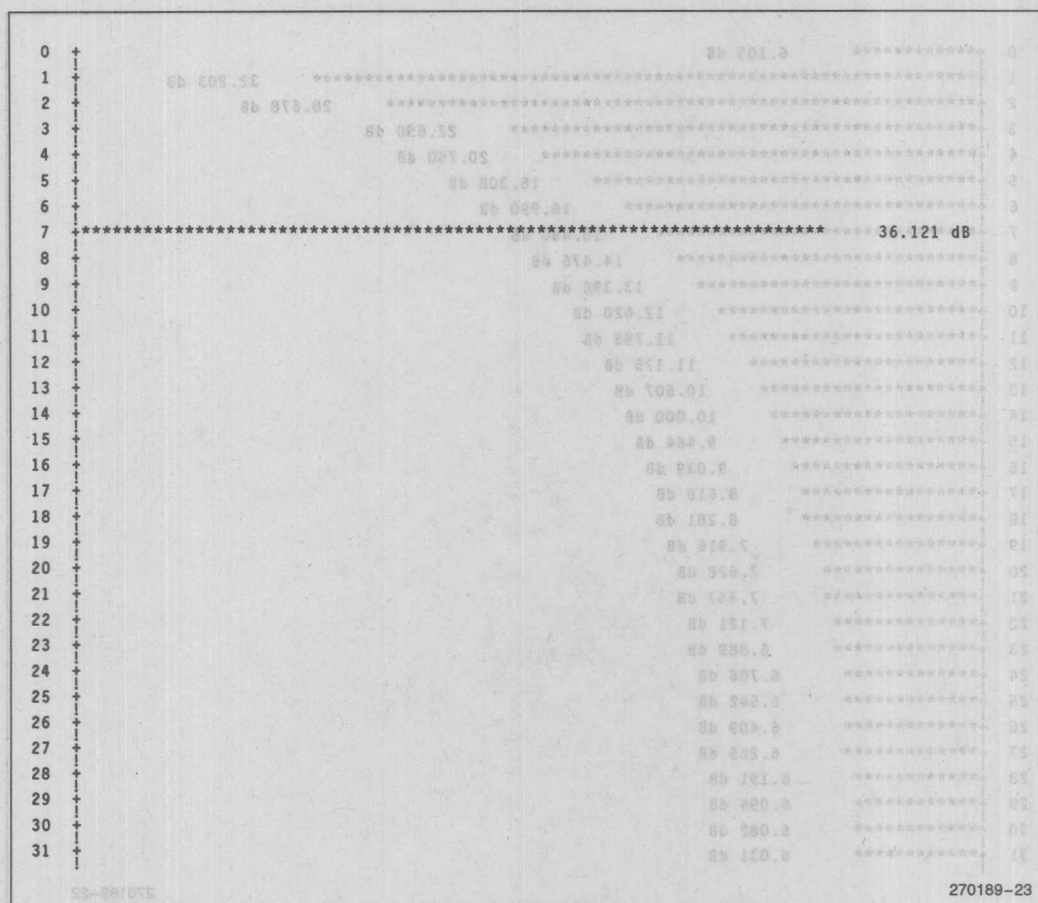
270189-20

Plot 1—Magnitude Plot of Squarewave

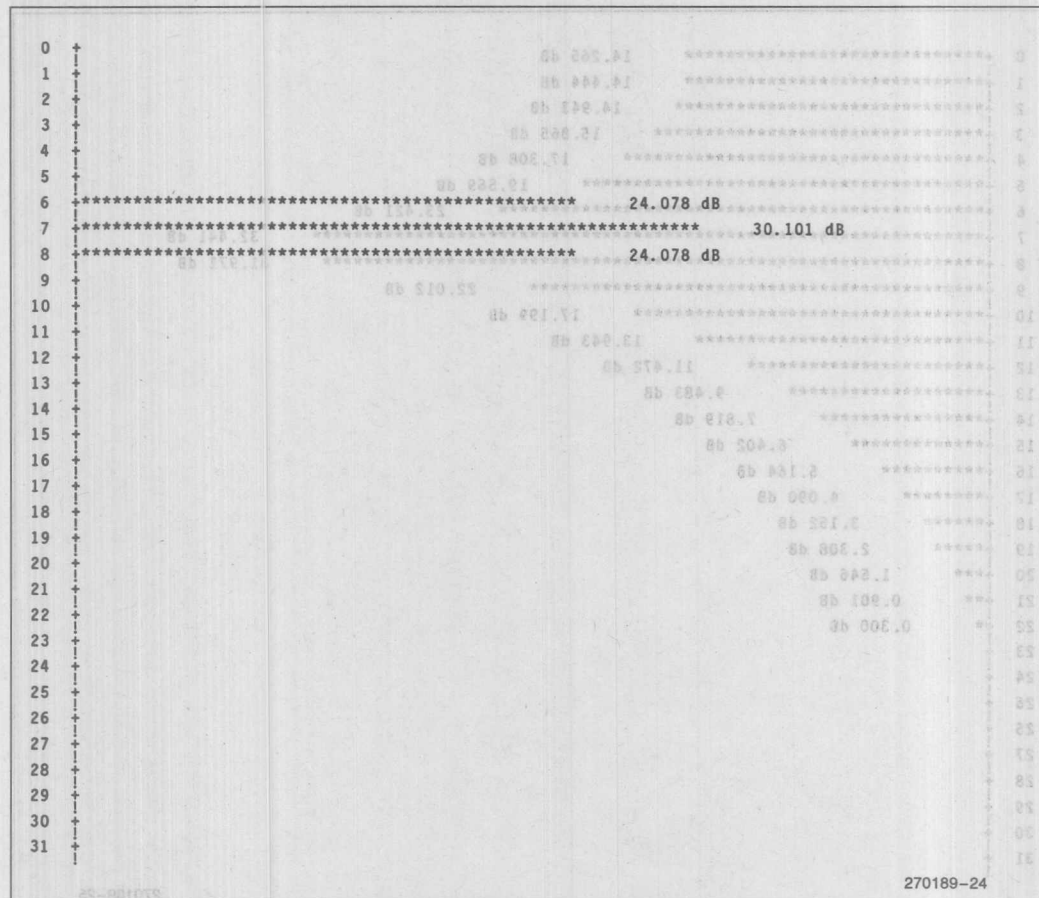


Plot 2—Decibel Plot of Squarewave

Plot 3—Plot of Squarewave with Window



Plot 4—Sin (7.0X) without Window

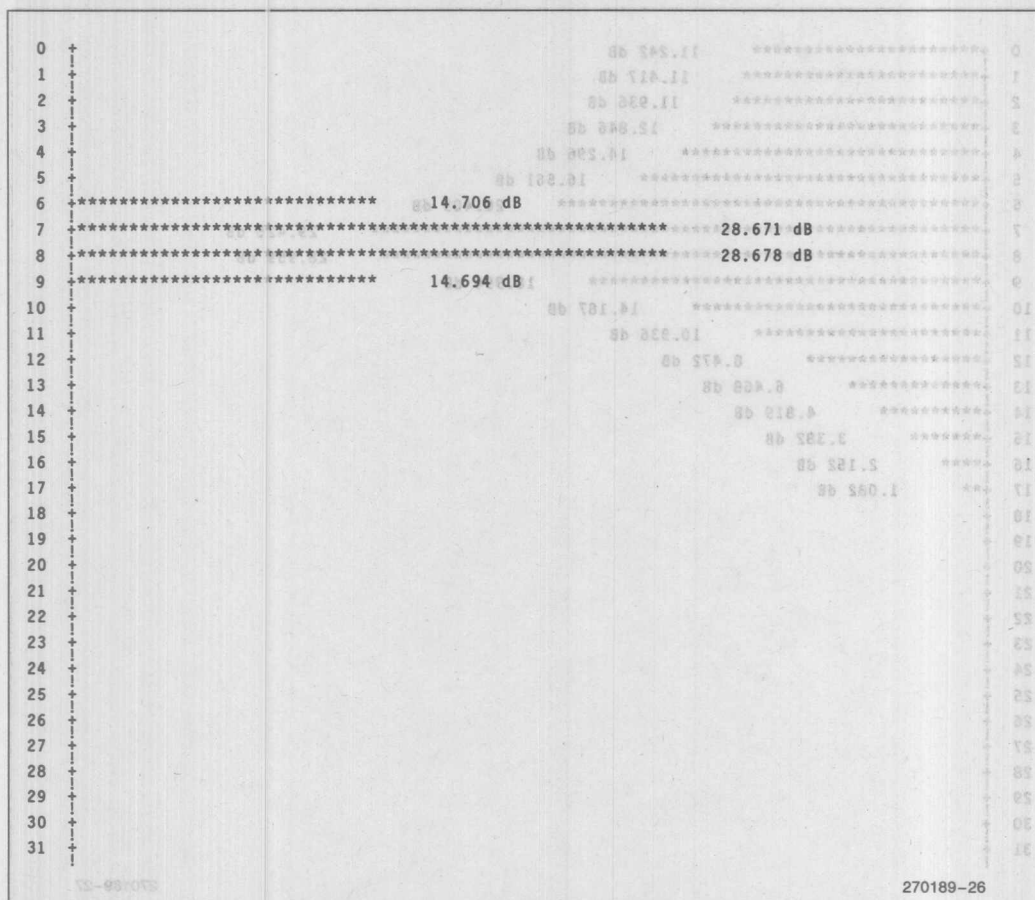


Plot 5—Sin (7.0X) with Window

0	*****	14.265 dB	
1	*****	14.444 dB	
2	*****	14.943 dB	
3	*****	15.865 dB	
4	*****	17.308 dB	
5	*****	19.569 dB	
6	*****	23.421 dB	
7	*****	32.441 dB	
8	*****	31.971 dB	
9	*****	22.012 dB	
10	*****	17.199 dB	
11	*****	13.943 dB	
12	*****	11.472 dB	
13	*****	9.483 dB	
14	*****	7.819 dB	
15	*****	6.402 dB	
16	*****	5.164 dB	
17	*****	4.090 dB	
18	*****	3.152 dB	
19	*****	2.308 dB	
20	***	1.546 dB	
21	**	0.901 dB	
22	*	0.300 dB	
23	+		
24	+		
25	+		
26	+		
27	+		
28	+		
29	+		
30	+		
31	+		

270189-25

Plot 6—Sin (7.5X) without Window

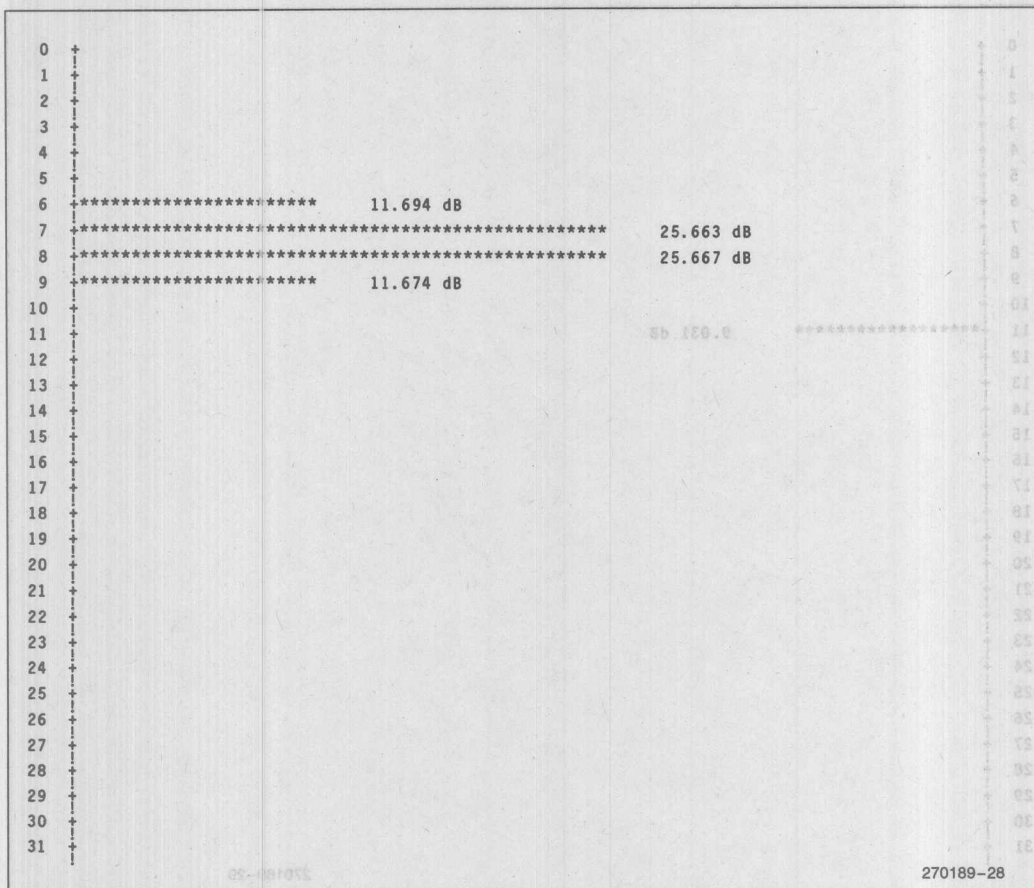


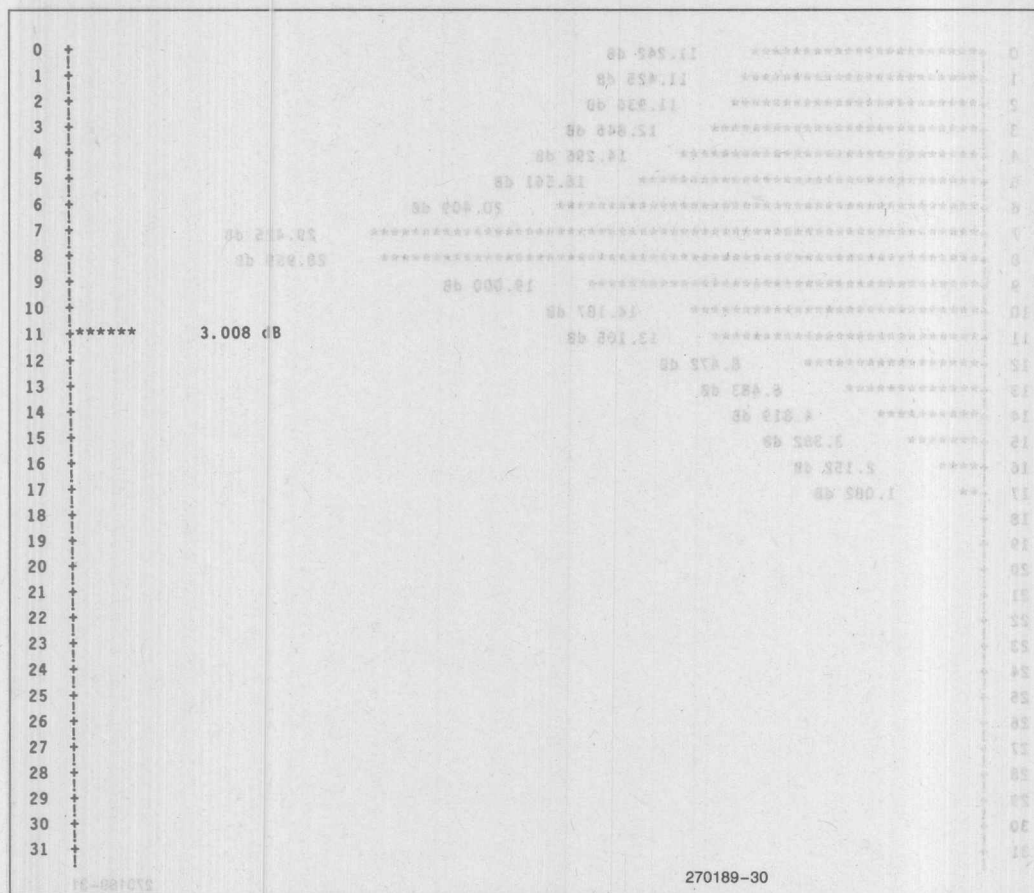
0	*****	11.242 dB	0
1	*****	11.417 dB	1
2	*****	11.936 dB	2
3	*****	12.846 dB	3
4	*****	14.296 dB	4
5	*****	16.561 dB	5
6	*****	20.409 dB	6
7	*****	29.425 dB	7
8	*****	28.959 dB	8
9	*****	18.994 dB	9
10	*****	14.187 dB	10
11	*****	10.936 dB	11
12	*****	8.472 dB	12
13	*****	6.468 dB	13
14	*****	4.819 dB	14
15	*****	3.382 dB	15
16	****	2.152 dB	16
17	**	1.082 dB	17
18	+		18
19	+		19
20	+		20
21	+		21
22	+		22
23	+		23
24	+		24
25	+		25
26	+		26
27	+		27
28	+		28
29	+		29
30	+		30
31	+		31

85-081015

270189-27

Plot 8— $0.707 \cdot \sin(7.5X)$ without Window





```

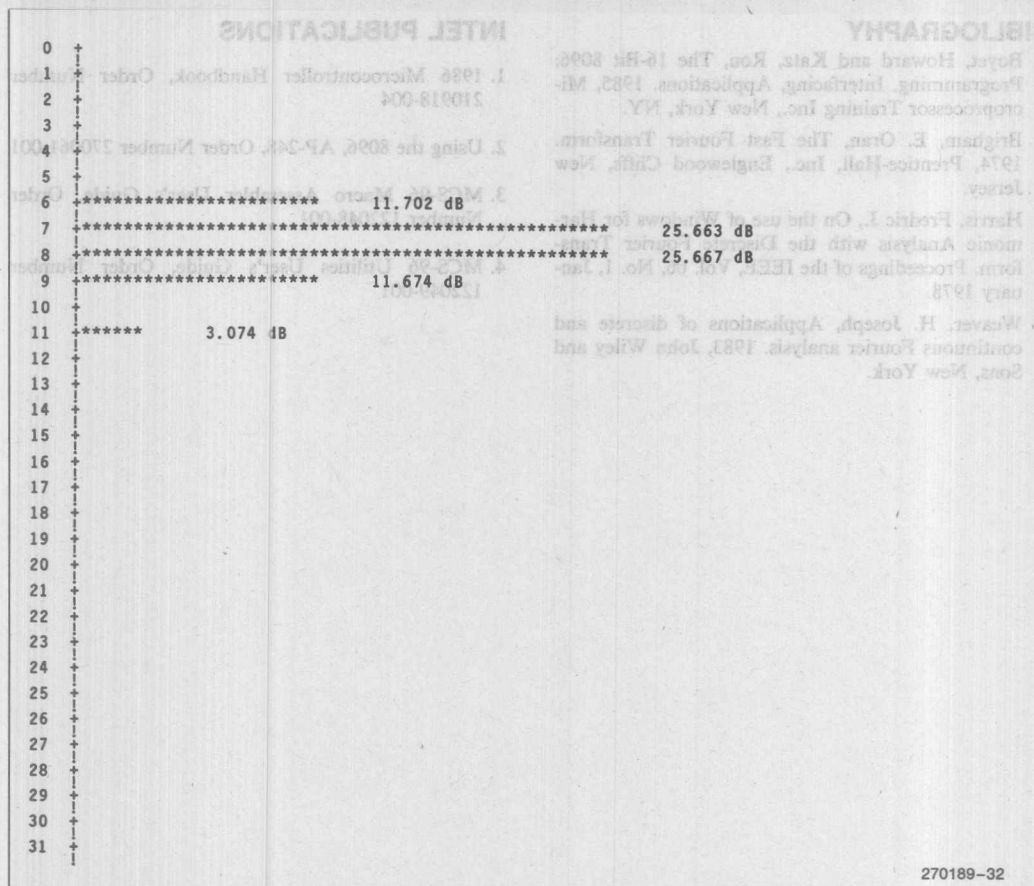
0 ***** 11.242 dB
1 ***** 11.425 dB
2 ***** 11.936 dB
3 ***** 12.846 dB
4 ***** 14.296 dB
5 ***** 16.561 dB
6 ***** 20.409 dB
7 ***** 29.425 dB
8 ***** 28.959 dB
9 ***** 19.000 dB
10 ***** 14.187 dB
11 ***** 13.105 dB
12 ***** 8.472 dB
13 ***** 6.483 dB
14 ***** 4.819 dB
15 ***** 3.382 dB
16 ***** 2.152 dB
17 ** 1.082 dB
18 +
19 +
20 +
21 +
22 +
23 +
24 +
25 +
26 +
27 +
28 +
29 +
30 +
31 +

```

CC-801013

270189-31

Plot 12— $0.707 (\sin (7.5X) + \frac{1}{16} \sin (11X))$ without Window



Plot 13—0.707 (Sin (7.5X) + $\frac{1}{16}$ Sin (11X)) with Window

BIBLIOGRAPHY

1. Boyet, Howard and Katz, Ron, The 16-Bit 8096: Programming, Interfacing, Applications. 1985, Microprocessor Training Inc., New York, NY.
2. Brigham, E. Oran, The Fast Fourier Transform. 1974, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
3. Harris, Fredric J., On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform. Proceedings of the IEEE, Vol. 66, No. 1, January 1978.
4. Weaver, H. Joseph, Applications of discrete and continuous Fourier analysis. 1983, John Wiley and Sons, New York.

INTEL PUBLICATIONS

1. 1986 Microcontroller Handbook, Order Number 210918-004
2. Using the 8096, AP-248, Order Number 270061-001
3. MCS-96 Macro Assembler User's Guide, Order Number 122048-001
4. MCS-96 Utilities User's Guide, Order Number 122049-001



APPLICATION BRIEF

AB-32

PAGE

6-183

6-183

CONTENTS

80C196 OVERVIEW

DESIGN GUIDELINES

UPGRADE PATH FROM
8096-90 TO 8096BH TO
80C196

April 1989

Upgrade Path from 8096-90 to 8096BH to 80C196

6

**UPGRADE PATH FROM
8096-90 TO 8096BH TO
80C196****CONTENTS**

PAGE

80C196 OVERVIEW 6-186

DESIGN GUIDELINES 6-186

April 1988

Upgrade Path from 8096-90 to
8096BH to 80C196

6

Converting applications that use an 8X9X-90 to use an 8X9XBH requires consideration of a few of the BH enhancements. Descriptions of each of the differences between the -90 and the BH follow, along with a discussion of the implications of the change.

BHE and INST are latched: The bus control signals $\overline{\text{BHE}}$ and $\overline{\text{INST}}$ are valid throughout the bus cycle on 8X9XBH devices. ON -90 devices, these signals need to be latched on the falling edge of ALE.

Byte Read following $\overline{\text{RESET}}$ rising: The bus control and buswidth options of 8X9XBH devices are selected by configuration of the chip immediately following the rising edge of $\overline{\text{RESET}}$. During the usual 10 state reset sequence, BH parts will perform a byte read of location 2018H to acquire configuration information prior to fetching the first opcode at location 2080H. The 8X9X-90 does not perform this read.

ALE is high while in reset: The ALE/ $\overline{\text{ADV}}$ pin of the 8X9XBH is driven high while the $\overline{\text{RESET}}$ pin is held low. On -90 devices, ALE is driven low while in $\overline{\text{RESET}}$. Circuits which rely on the state of ALE while $\overline{\text{RESET}}$ is low must be modified. The reset state of ALE was changed to enable implementation of the Chip Configuration Byte read from external memory following the rising edge of $\overline{\text{RESET}}$.

EA is latched on $\overline{\text{RESET}}$ rising: The 8X9XBH latches the value of $\overline{\text{EA}}$ on the rising edge of $\overline{\text{RESET}}$. On -90 devices, EA was not latched and could be changed without placing the part in $\overline{\text{RESET}}$. This change was necessary to enhance ROM/EPROM security. Circuits that rely on $\overline{\text{EA}}$ not being latched must be modified.

A/D speed increased: The 8X95BH and 8X97BH A/D converters complete conversion in 88 state times. On -90 devices with A/D converters, a conversion takes 168 state times. This translates in an increased conversion speed from 42 μs on -90 parts to 22 μs on BH parts running at 12MHz. Software that relies upon the speed of conversion for timing must be changed. It is also recommended that MCS-96 software be written so as to not be impacted by further changes in A/D conversion speed.

Sample/Hold on A/D: The 8X95BH and 8X97BH have a sample/hold on the input of the A/D converter. 8X9X-90 devices with A/D converters do not have sample/hold circuitry. External analog circuitry which also includes a sample/hold must provide a settled analog input within the first four state times of 8X9XBH conversion.

Duplicate Fetches: The 8X9XBH bus controller was made more aggressive when it comes to instruction fetches in order to minimize the execution speed degra-

dation of using an 8-bit bus. As a result, instruction fetches over a 16-bit bus sometimes occur when there is no space in the prefetch queue to store the fetched opcodes. This requires another instruction fetch from the same address when space in the prefetch queue opens up.

To the external system, these occurrences appear as duplicate instruction fetches. An estimated 10 percent of all instruction fetches will be "duplicates", while overall bus loading will be approximately 65 to 70 percent, compared to an 8X9X-90 bus loading of approximately 55 to 60 percent. Execution speed is not impacted by a duplicate fetch.

Write Pulse Width: The 8X9XBH 16-bit bus write pulse width is one T_{osc} longer than on the 8X9X-90, thus allowing slower memories and peripherals to be used. In order to widen the $\overline{\text{WR}}$ pulse width, the time between the end of $\overline{\text{WR}}$ and the next ALE was reduced by T_{osc} . Note that the signals $\overline{\text{WRL}}$, $\overline{\text{WRH}}$, and $\overline{\text{WR}}$ with an 8-bit bus are still the same width as on -90 parts.

Vpp Replaces VBB: Vpp is the programming pin for EPROM devices. Systems that have this connected through a capacitor to ANGND (required on 8X9X-90 parts) do not need to change. ANGND must be held nominally at the same potential as V_{SS} , and Vpp must NOT be connected to V_{CC} . High voltage must NEVER be placed on the Vpp pin of a ROM device.

While there is almost no reason to do so, an application should not attempt to execute with the EA pin at logic zero and V_{CC} at 5.5 V_{DC} on an 879XBH EPROM device. Additionally, the design should always begin the "out of $\overline{\text{RESET}}$ " code execution from the internal EPROM, immediately after the power-on sequence.

Reserved location warning: Intel reserved addresses can not be used by applications which use 8X9XBH internal ROM/EPROM. The data read from a reserved location is not guaranteed, and a write to any reserved location could cause unpredictable results. When attempting to program Intel Reserved addresses, the data must be OFFFFFH to ensure a harmless result.

Intel Reserved locations, when mapped to external memory, must be filled with OFFFFFH to ensure compatibility with future parts.

A positive transition on NMI: The 8X9XBH does not clear the Watchdog Timer. The 8X9X-90 does clear the WDT on a positive transition of NMI, and both part vector to external address 0000H.

The following is the latest information on upgrading a NMOS 8096 to a CHMOS 80C196.

The chip which is the CMOS 80C196 replacement is designated the 80C196. The part can be configured to be pin compatible with the 8096, but because of the process change and other enhancements, it may not be plug compatible in some designs. This is to say that you will not be able to arbitrarily swap out a NMOS 8096 and replace it with the 80C196. However, if a few rules are followed the changes required will be almost painless.

80C196 OVERVIEW

First, some background on the 80C196 is needed. The opcode set is a true superset of the 8096, but some enhancements have been made to the peripherals and timings. The crystal is divided by 2 on the 80C196, instead of 3, as on the 8096. This means that the 80C196 running at 8 MHz will have a 250 ns state time, Just like an 8096 running at 12 MHz.

An 80C196 running at 8 MHz will emulate an 8096 at 12 MHz except that some of the instructions and peripherals will operate faster. The instructions which will be speeded up include mul, div, interrupt, call, ret, and jumps. The serial port will require a different baud value and the A to D may not run at exactly the same speed. This means that timing loops which measure instruction speed or A to D completion speed may have to be modified. The bus timings, while not nanosecond for nanosecond compatible, will work in most systems.

DESIGN GUIDELINES

1. Do not use undefined register areas for storage or depend on them to return a specific value if it is not stated in the Embedded Controller. Undefined registers and locations on this, or any other, part should be considered off-limits and reserved for development systems, testing or future use.
2. Do not base timings loops on instruction execution times, as some instructions may execute faster on the 80C196 than on the 8096, even when the 80C196 is slowed down to 8 MHz, its 8096 compatible rate. Counter-type loops should be initialized with values that can easily be changed at compile time.

A to D completions, flag settings, etc. This is for the same reason as above; some of these responses may be slightly different from those on the 8096. Timer 1 is provided for critical timings. With an 8 MHz crystal, it will increment every 2 microseconds, just as an 8096 running at 12 MHz.

4. The serial port baud register values should be easily changeable at compile time. Since the serial port is now capable of running at a higher frequency, a different baud rate value will be needed.
5. The circuitry interfacing to the chip should be capable of interfacing to the 80C196. The I/O lines on 80C196 will look a lot like those on the 80C51.
6. The $\overline{\text{BHE}}/\overline{\text{WRH}}$ signal in eight bit and write strobe mode will go low for odd byte transfers and high for even byte transfers. The $\overline{\text{WR}}/\overline{\text{WRL}}$ signal will go low for odd byte transfers and high for even byte transfers. Normally, the $\overline{\text{WR}}/\overline{\text{WRL}}$ signal should go low for odd and even byte transfers since transfers are on the low byte of the data bus.
7. PUSH and POP operations addressed relative to the stack pointer work differently on the 80C196 than on the 8096. On the 8096, the address is calculated based on the un-updated stack pointer value, on the 80C196, the address is calculated based on the updated value. The only operations effected are: PUSH xx[sp], PUSH [sp], PUSH sp, POP xx[sp], POP [sp], POP sp.
8. The V_{DD} pin on the 8X9X parts is now the CDE (Clock Detect Enable) pin on the 80C196. When tied high, CDE enables a clock speed sensor and will reset the part if the Xtall frequency drops below a few hundred KHz. While this is perfect for most production boards, it may be desirable to have a jumper option on this function for evaluation boards.



APPLICATION BRIEF

AB-33

PAGE

6-189	THE 286K SYSTEM
6-189	Hardware
6-189	Software
6-190	THE 544K SYSTEM
6-190	Hardware
6-190	Software
6-190	THE INST PIN
6-190	Instruction Fetches
6-190	Data Reads and Writes

April 1989

Memory Expansion for the 8096

6

DOUG YODER
ECO APPLICATIONS ENGINEER

Order Number: 270522-001

MEMORY EXPANSION FOR THE 8096

CONTENTS

PAGE

THE 256K SYSTEM	6-189
Hardware	6-189
Software	6-189
THE 544K SYSTEM	6-190
Hardware	6-190
Software	6-190
THE INST PIN	6-190
Instruction Fetches	6-190
Data Reads and Writes	6-190

April 1988

Memory Expansion for the 8096

DOUG YODER
ECO APPLICATIONS ENGINEER

Order Number: 370523-001

This Application Brief presents two examples of a paging scheme for the 8096, allowing either 256K bytes of total memory, or 544K bytes of total memory. Both systems utilize PORT1 as the output for the upper address lines. Because Interrupt vectors, and other critical sections of code must always be present, addresses 0-7FFFH always refer to the same main page. The PORT1 upper addresses only affect addresses 8000-FFFFH, by slapping several 32K pages in and out.

THE 256K SYSTEM

Hardware

The hardware for the 256K system (see Figures 4 & 5, an example with 128K ROM and 128K RAM) utilizes a 74LS157 quad 2 to 1 multiplexer. The enable pin of the 74LS157 is tied to the inverted A15 signal, which is the latched addr/data 15 (AD15) signal from the 96. In this way, when A15 is low, the 74LS157 is disabled and all its outputs are low. Particularly, MA17 is low, which selects the 27512 and deselects the rams. Also, MA15 and MA16 are low, which guarantee that addresses 0-7FFFH of the 27512 are accessed.

When A15 is high, the 74LS157 is enabled to pass MA15 - MA17 values. The bank select pin of the 74LS157 is connected to the INST pin of the 96. When the INST pin is high, for a code access, INSTA15 - INSTA17 (PORT1.0 - PORT1.2) are used. When INST is low, for a data read or write, DATAA15 - DATAA17 (PORT1.3 - PORT 1.5) are used. This allows for the use of separate pages for code and data without having to change the upper address lines each time. Also, it is possible to select a ROM page for a data table, or load a RAM page with executable code downloaded from another source. PORT1.6 and PORT1.7 can still be used as I/O ports. If a -90 part were used, the INST pin would need to be latched since it is only valid during the address output on the bus pins.

This system was designed to get the maximum amount of memory with a minimum amount of hardware. The

amount of ROM and RAM was picked arbitrarily, and could be reconfigured in various ways, however, this may require slight modifications or additions to the decoder circuitry. This setup has a main page at addresses 0-7FFFH, and upper pages 1-7 at addresses 8000-FFFFH. Note that upper page 0 is the same as the main page. The WRL and WRH feature of the BH part was used to allow for byte writes to RAM. If the -90 part were to be used, additional logic would be necessary to generate these signals from WR and BHE.

The RAM chips utilized were NEC uPD43256-15 32K x 8 static rams with an access time of 150ns. The ROMs were Intel 27512 64K x 8 EPROMs with an access time of 200ns. The decoder circuitry used was entirely LS TTL. Using an 8097BH running at 10MHz, there was ample time for address decoding and memory access. Timing analysis showed that 12MHz operation would also be accommodated easily. If slower memories are used, further analysis would be necessary. Also, it would be possible to switch to S TTL to greatly decrease the decoding response time.

Software

When using this system there are several things to keep in mind when preparing the software.

Since ASM96 will only allow addresses from 0-FFFFH, it is necessary to generate each page of code in a separate file. These pages should not be linked together, but rather should each be used to program the proper section of the EPROM associated with that page. The main page routine should be coded with addresses from 0-7FFFH, and each of the upper pages should be coded with addresses from 8000-FFFFH. Because linking is not possible, each module should contain a table of constants which defines the symbols used in other modules. These values are easily obtained from the listing file, which can be created using zeros in the table the first time. The addresses of the pages in a 27512 after splitting low and high bytes into 2 EPROMs are shown in Figure 1.

EPROM LOCATION U5		EPROM LOCATION U6		RAM LOCATION U7		RAM LOCATION U8		RAM LOCATION U9		RAM LOCATION U10	
0H	MAIN PAGE LOW	0H	MAIN PAGE HIGH	0H	PAGE4 LOW BYTES	0H	PAGE4 HIGH BYTES	0H	PAGE6 LOW BYTES	0H	PAGE8 HIGH BYTES
3FFFH		3FFFH		3FFFH		3FFFH		3FFFH		3FFFH	
4000H	PAGE1 LOW BYTES	4000H	PAGE1 HIGH BYTES	4000H	PAGE5 LOW BYTES	4000H	PAGE5 HIGH BYTES	4000H	PAGE7 LOW BYTES	4000H	PAGE7 HIGH BYTES
7FFFH		7FFFH		7FFFH		7FFFH		7FFFH		7FFFH	
8000H	PAGE2 LOW BYTES	8000H	PAGE2 HIGH BYTES								
BFFFH		BFFFH									
C000H	PAGE3 LOW BYTES	C000H	PAGE3 HIGH BYTES								
FFFFH		FFFFH									

Figure 1. The Current System

EPROM LOCATION U5		EPROM LOCATION U6		EPROM LOCATION U7		EPROM LOCATION U8	
0H	MAIN PAGE LOW	0H	MAIN PAGE HIGH	0H	PAGE4 LOW BYTES	0H	PAGE4 HIGH BYTES
3FFFH		3FFFH		3FFFH		3FFFH	
4000H	PAGE1 LOW BYTES	4000H	PAGE1 HIGH BYTES	4000H	PAGE5 LOW BYTES	4000H	PAGE5 HIGH BYTES
7FFFH		7FFFH		7FFFH		7FFFH	
8000H	PAGE2 LOW BYTES	8000H	PAGE2 HIGH BYTES	8000H	PAGE6 LOW BYTES	8000H	PAGE6 HIGH BYTES
BFFFH		BFFFH		BFFFH		BFFFH	
C000H	PAGE3 LOW BYTES	C000H	PAGE3 HIGH BYTES	C000H	PAGE7 LOW BYTES	C000H	PAGE7 HIGH BYTES
FFFFH		FFFFH		FFFFH		FFFFH	

Figure 2. A System Using all EPROMS and no RAM

All changes to the upper instruction addresses of PORT1 must be made by code located in the main page. A listing of subroutines for use in the main page, and a listing of macros for use in all pages is provided. By invoking one of these macros the programmer can easily transfer from one page to another, or select a new data page. The subroutines should not be called directly, they should be entered by using the appropriate macro. The subroutines should be located at the addresses specified, otherwise the macros must be changed as they are written to call an absolute address in the main page. Also, any hardware changes may render the software inoperative.

Because the WRL-WRH feature of the 96BH is used, the correct Chip Configuration Register value of 0FBH must be loaded into the ROMs at address 2018H. This is done in the main code file with the following statements:

```
CSEG AT 2018H
```

```
CCR: DCB 0FBH ;VALUE FOR CHIP  
CONFIGURATION REGISTER
```

Finally, it is necessary to initialize the DATA address at the start of the program this can be done using the NEW_DATA_PAGE MACRO.

THE 544K SYSTEM

Hardware

The hardware for the 544K system (see Figures 6 & 7, an example with 288K ROM and 256K RAM) has some slight changes from the 256K system.

First, all pins of PORT1 are now in use as address lines. This allows for PORT1 to select 16 pages of memory, with a different address for instructions or data.

Second, 27128 16K x 8 EPROMS have been added for use as the main code page. In this system, the main page is physically separate from upper page 0. The 27128's are selected by A15 being low. The upper pages of memory are selected when A15 is high which enables the 74LS155 demultiplexer which is used for address decoding. When the 74LS155 is disabled, its outputs are all high, which disables all upper memories. The 74LS157 is enabled all the time, to speed up address decoding, as its outputs do not matter when the 74LS155 is disabled.

Software

All rules for the 256K system apply to the 544K system, except that the main page no longer overlaps page 0. However, because all of PORT1 is now in use, different macros and subroutines must now be used. These have been included also.

THE INST PIN

The instruction pin has been verified to work correctly on the 8X9X- 90, 8X9XBH, and the 80C196. The functionality of the INST pin is as follows.

Instruction Fetches

The INST pin is high during an external memory read indicating the read is an instruction fetch. This includes immediate data reads since the data is embedded in the code.

Data Reads and Writes

The INST is low during an external memory read or write indicating the bus cycle is a data cycle. This would be indirect and indexed instructions which are directed at external memory.

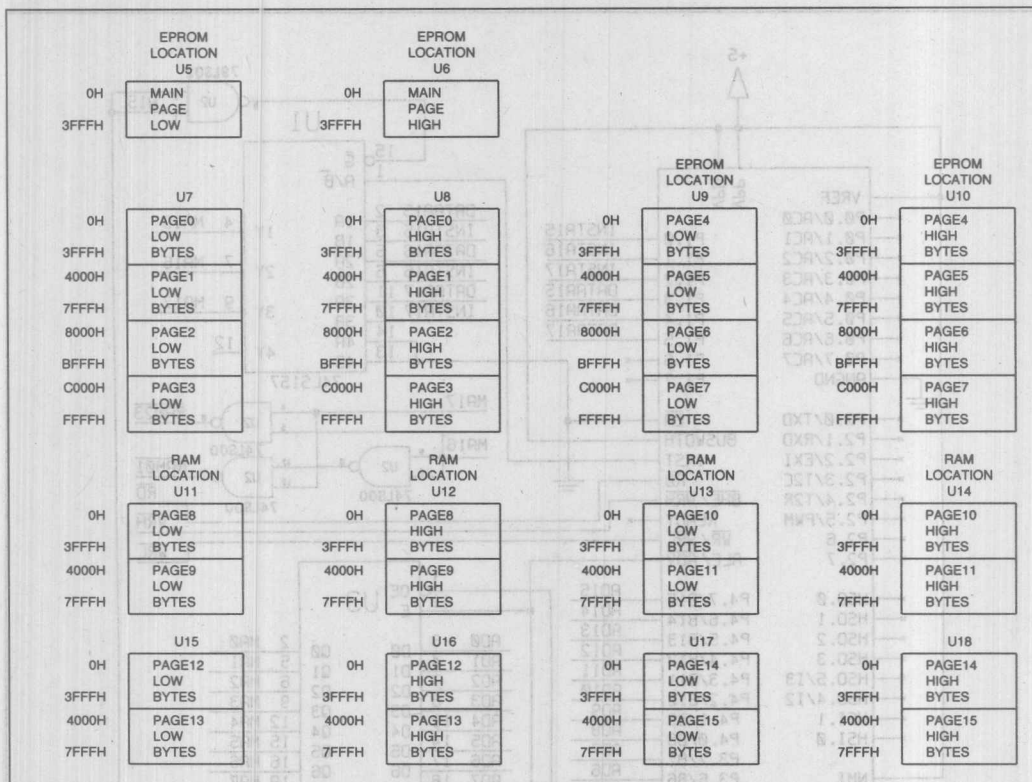


Figure 3. The 544K Memory Map

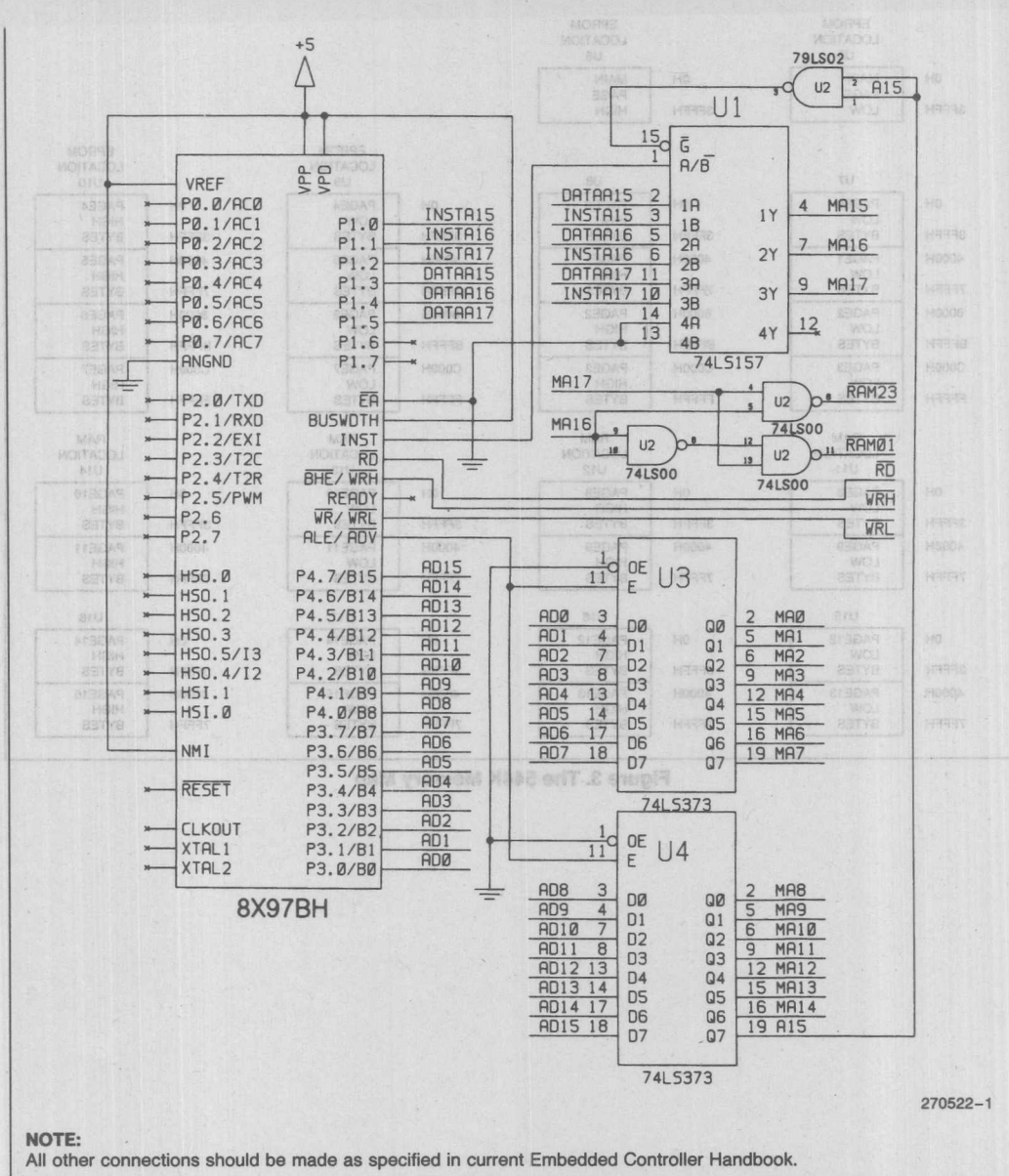


Figure 4. 128K ROM + 128K RAM Memory

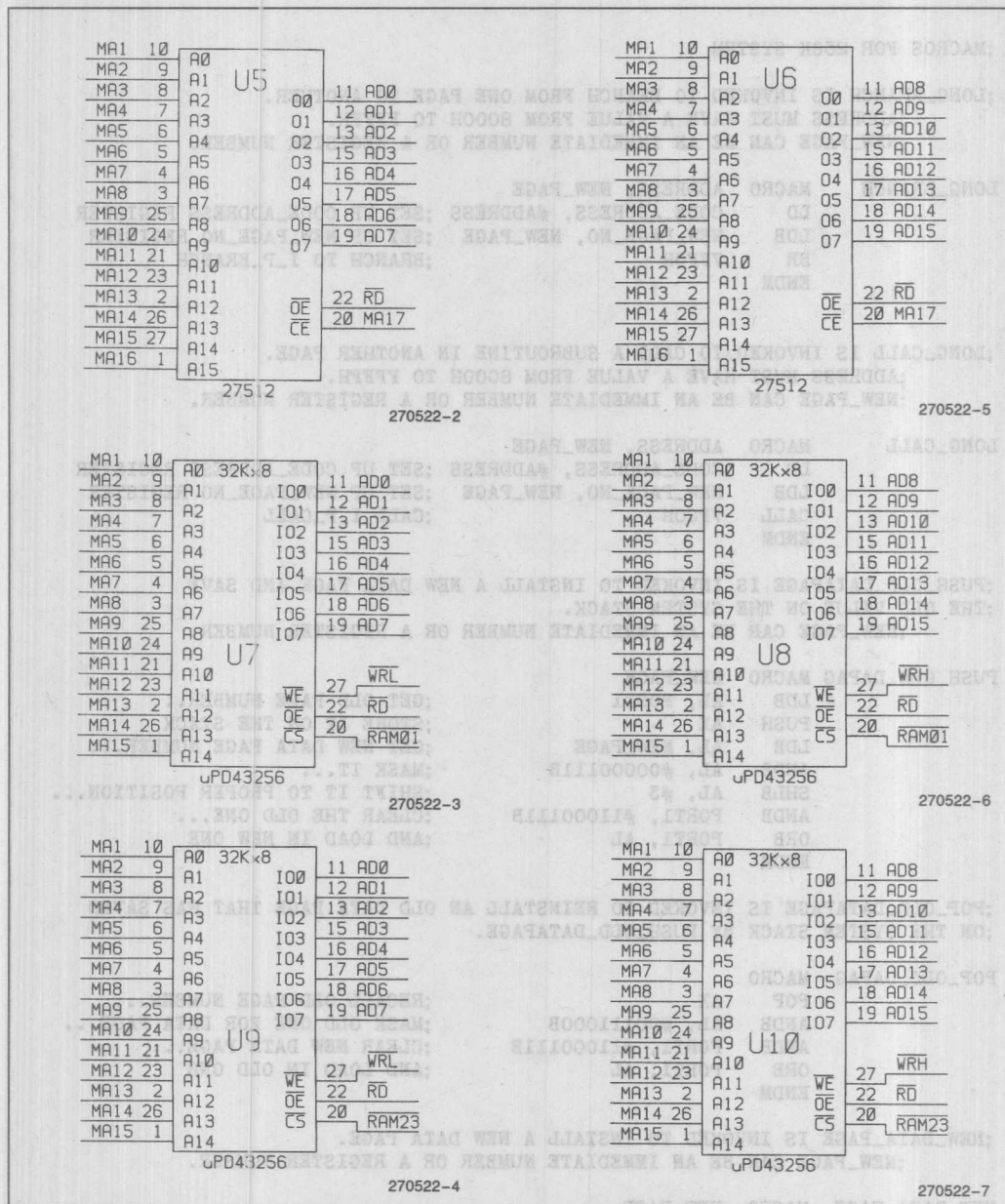


Figure 5. 128K ROM + 128K RAM Memory

;MACROS FOR 256K SYSTEM

;LONG_BRANCH IS INVOKED TO BRANCH FROM ONE PAGE TO ANOTHER.

;ADDRESS MUST HAVE A VALUE FROM 8000H TO FFFFH.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
LONG_BRANCH    MACRO    ADDRESS, NEW_PAGE
                LD      CODE_ADDRESS, #ADDRESS ;SET UP CODE_ADDRESS REGISTER
                LDB     NEW_PAGE_NO, NEW_PAGE  ;SET UP NEW_PAGE_NO REGISTER
                BR      7FF0H                  ;BRANCH TO I_P_BRANCH
            ENDM
```

;LONG_CALL IS INVOKED TO CALL A SUBROUTINE IN ANOTHER PAGE.

;ADDRESS MUST HAVE A VALUE FROM 8000H TO FFFFH.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
LONG_CALL      MACRO    ADDRESS, NEW_PAGE
                LD      CODE_ADDRESS, #ADDRESS ;SET UP CODE_ADDRESS REGISTER
                LDB     NEW_PAGE_NO, NEW_PAGE  ;SET UP NEW_PAGE_NO REGISTER
                CALL    7FC0H                  ;CALL I_P_CALL
            ENDM
```

;PUSH_OLD_DATAPAGE IS INVOKED TO INSTALL A NEW DATA PAGE AND SAVE
;THE OLD VALUE ON THE SYSTEM STACK.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
PUSH_OLD_DAPAG MACRO    NEW_PAGE
                LDB     AL, PORT1              ;GET OLD PAGE NUMBER...
                PUSH    AX                    ;STORE IT ON THE STACK
                LDB     AL, NEW_PAGE          ;GET NEW DATA PAGE NUMBER...
                ANDB    AL, #00000111B        ;MASK IT...
                SHLB    AL, #3                ;SHIFT IT TO PROPER POSITION...
                ANDB    PORT1, #11000111B     ;CLEAR THE OLD ONE...
                ORB     PORT1, AL             ;AND LOAD IN NEW ONE
            ENDM
```

;POP_OLD_DATAPAGE IS INVOKED TO REINSTALL AN OLD DATA PAGE THAT WAS SAVED
;ON THE SYSTEM STACK BY PUSH_OLD_DATAPAGE.

```
POP_OLD_DAPAG  MACRO
                POP     AX                    ;RECALL OLD PAGE NUMBER...
                ANDB    AL, #00111000B        ;MASK OLD ONE FOR DATA PAGE...
                ANDB    PORT1, #11000111B     ;CLEAR NEW DATA PAGE...
                ORB     PORT1, AL             ;AND LOAD IN OLD ONE
            ENDM
```

;NEW_DATA_PAGE IS INVOKED TO INSTALL A NEW DATA PAGE.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
NEW_DATA_PAGE  MACRO    NEW_PAGE
                LDB     AL, NEW_PAGE          ;GET NEW DATA PAGE NUMBER...
                ANDB    AL, #00000111B        ;MASK IT...
                SHLB    AL, #3                ;SHIFT IT TO PROPER POSITION...
                ANDB    PORT1, #11000111B     ;CLEAR THE OLD ONE...
                ORB     PORT1, AL             ;AND LOAD IN NEW ONE
            ENDM
```


;SUBROUTINES FOR 256K SYSTEM

CSEG AT 7FC0H

;SUBROUTINE: I_P_CALL

; THIS SUBROUTINE ALLOWS FOR THE CALLING OF SUBROUTINES LOCATED IN
; A DIFFERENT PAGE OF MEMORY.

; PARAMETERS: CODE_ADDRESS, NEW_PAGE_NO

; SUBROUTINES: ANY THAT ARE REQUESTED.

```

I_P_CALL:      IDB      AL, PORT1      ;GET OLD PAGE NUMBER...
               FUSH     AX              ;STORE IT ON THE STACK
               ANDB     PORT1, #11111000B ;CLEAR OLD INST PAGE...
               ANDB     NEW_PAGE_NO, #00000111B ;MASK NEW ONE...
               ORB      PORT1, NEW_PAGE_NO ;AND LOAD IT IN
               FUSH     #I_P_RETURN     ;SAVE RETURN ADDRESS...
               ER        [CODE_ADDRESS] ;CALL REQUESTED ROUTINE

I_P_RETURN:    POP      AX              ;RECALL OLD PAGE NUMBER...
               ANDB     PORT1, #11111000B ;CLEAR NEW INST PAGE...
               ANDB     AL, #00000111B ;MASK OLD ONE...
               ORB      PORT1, AL        ;AND LOAD IT IN
               RET                     ;RETURN TO CALLING ROUTINE

```

CSEG AT 7FF0H

;SUBROUTINE: I_P_BRANCH

; THIS SUBROUTINE ALLOWS FOR BRANCHING TO LOCATIONS IN A DIFFERENT
; PAGE OF MEMORY.

; PARAMETERS: CODE_ADDRESS, NEW_PAGE_NO

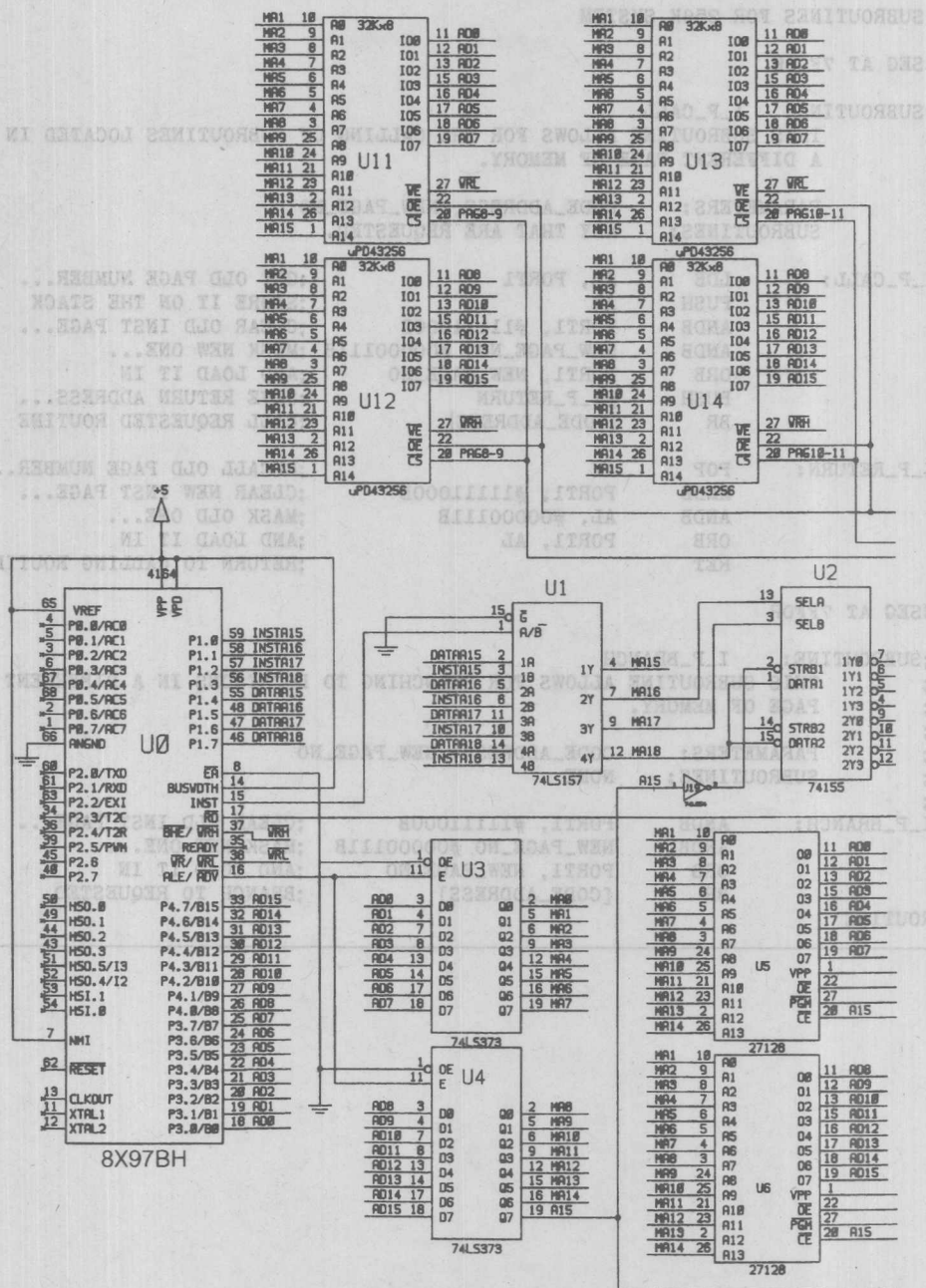
; SUBROUTINES: NONE

```

I_P_BRANCH:    ANDB     PORT1, #11111000B ;CLEAR OLD INST PAGE...
               ANDB     NEW_PAGE_NO #00000111B ;MASK NEW ONE...
               ORB      PORT1, NEW_PAGE_NO ;AND LOAD IT IN
               BR        [CODE_ADDRESS] ;BRANCH TO REQUESTED

```

ROUTINE



270522-8

NOTE:

All other connections should be made as specified in current Embedded Controller Handbook.

Figure 6. 288K ROM + 256K RAM Memory

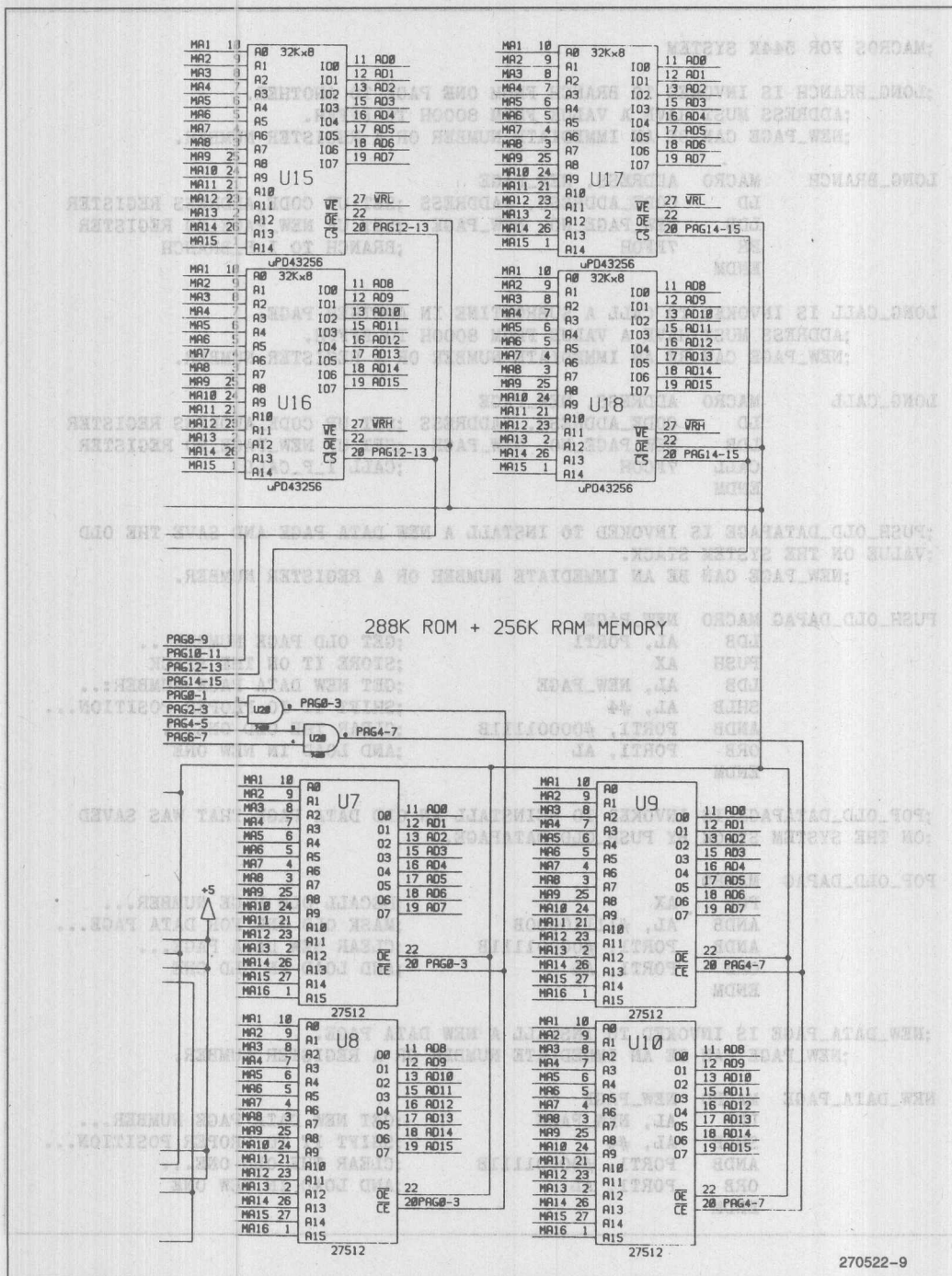


Figure 7. 288K ROM + 256K RAM Memory

;MACROS FOR 544K SYSTEM

;LONG_BRANCH IS INVOKED TO BRANCH FROM ONE PAGE TO ANOTHER.

;ADDRESS MUST HAVE A VALUE FROM 8000H TO FFFFH.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
LONG_BRANCH  MACRO  ADDRESS, NEW_PAGE
               LD      CODE_ADDRESS, #ADDRESS ;SET UP CODE_ADDRESS REGISTER
               LDB      NEW_PAGE_NO, NEW_PAGE ;SET UP NEW_PAGE_NO REGISTER
               BR       7FF0H ;BRANCH TO I_P_BRANCH
               ENDM
```

LONG_CALL IS INVOKED TO CALL A SUBROUTINE IN ANOTHER PAGE.

;ADDRESS MUST HAVE A VALUE FROM 8000H TO FFFFH.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
LONG_CALL    MACRO  ADDRESS, NEW_PAGE
               LD      CODE_ADDRESS, #ADDRESS ;SET UP CODE_ADDRESS REGISTER
               LDB      NEW_PAGE_NO, NEW_PAGE ;SET UP NEW_PAGE_NO REGISTER
               CALL     7FC0H ;CALL I_P_CALL
               ENDM
```

;PUSH_OLD_DATAPAGE IS INVOKED TO INSTALL A NEW DATA PAGE AND SAVE THE OLD
;VALUE ON THE SYSTEM STACK.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
PUSH_OLD_DAPAG MACRO  NEW_PAGE
               LDB      AL, PORT1 ;GET OLD PAGE NUMBER...
               PUSH     AX ;STORE IT ON THE STACK
               LDB      AL, NEW_PAGE ;GET NEW DATA PAGE NUMBER...
               SHLB     AL, #4 ;SHIFT IT TO PROPER POSITION...
               ANDB     PORT1, #00001111B ;CLEAR THE OLD ONE...
               ORB      PORT1, AL ;AND LOAD IN NEW ONE
               ENDM
```

;POP_OLD_DATAPAGE IS INVOKED TO REINSTALL AN OLD DATA PAGE THAT WAS SAVED
;ON THE SYSTEM STACK BY PUSH_OLD_DATAPAGE.

```
POP_OLD_DAPAG MACRO
               POP      AX ;RECALL OLD PAGE NUMBER...
               ANDB     AL, #11110000B ;MASK OLD ONE FOR DATA PAGE...
               ANDB     PORT1, #00001111B ;CLEAR NEW DATA PAGE...
               ORB      PORT1, AL ;AND LOAD IN OLD ONE
               ENDM
```

;NEW_DATA_PAGE IS INVOKED TO INSTALL A NEW DATA PAGE.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
NEW_DATA_PAGE MACRO  NEW_PAGE
               LDB      AL, NEW_PAGE ;GET NEW DATA PAGE NUMBER...
               SHLB     AL, #4 ;SHIFT IT TO PROPER POSITION...
               ANDB     PORT1, #00001111B ;CLEAR THE OLD ONE...
               ORB      PORT1, AL ;AND LOAD IN NEW ONE
               ENDM
```



```
;SUBROUTINES FOR 544K SYSTEM
```

```
CSEG AT 7FC0H
```

```
;SUBROUTINE: I_P_CALL
```

```
; THIS SUBROUTINE ALLOWS FOR THE CALLING OF SUBROUTINES LOCATED IN  
; A DIFFERENT PAGE OF MEMORY.
```

```
; PARAMETERS: CODE_ADDRESS, NEW_PAGE_NO  
; SUBROUTINES: ANY THAT ARE REQUESTED.
```

```
I_P_CALL: LDB AL, PORT1 ;GET OLD PAGE NUMBER...  
          FUSH AX ;STORE IT ON THE STACK  
          ANDB PORT1, #11110000B ;CLEAR OLD INST PAGE...  
          ANDB NEW_PAGE_NO, #00001111B ;MASK NEW ONE...  
          ORB PORT1, NEW_PAGE_NO ;AND LOAD IT IN  
          FUSH #I_P_RETURN ;SAVE RETURN ADDRESS...  
          ER [CODE_ADDRESS] ;CALL REQUESTED ROUTINE
```

```
I_P_RETURN: FOP AX ;RECALL OLD PAGE NUMBER...  
            ANDB PORT1, #11110000B ;CLEAR NEW INST PAGE...  
            ANDB AL, #00001111B ;MASK OLD ONE...  
            ORB PORT1, AL ;AND LOAD IT IN  
            FET ;RETURN TO CALLING ROUTINE
```

```
CSEG AT 7FF0H
```

```
;SUBROUTINE: I_P_BRANCH
```

```
; THIS SUBROUTINE ALLOWS FOR BRANCHING TO LOCATIONS IN A DIFFERENT  
; PAGE OF MEMORY.
```

```
; PARAMETERS: CODE_ADDRESS, NEW_PAGE_NO  
; SUBROUTINES: NONE
```

```
I_P_BRANCH: ANDB PORT1, #11110000B ;CLEAR OLD INST PAGE...  
            ANDB NEW_PAGE_NO, #00001111B ;MASK NEW ONE...  
            ORB PORT1, NEW_PAGE_NO ;AND LOAD IT IN  
            ER [CODE_ADDRESS] ;BRANCH TO REQUESTED ROUTINE
```



APPLICATION BRIEF

AB-34

April 1989

Integer Square Root Routine for the 8096

LIONEL SMITH
ECO APPLICATIONS ENGINEER

Order Number: 270523-001

The program module which follows is part of a collection of routines which perform integer and real arithmetic on a software implemented tagged stack. The top element of the stack is called TOS and is in fixed location in the register file. Since the square root operation only involves TOS, further details of the stack structure are not shown.

Comments

Average Execution Time	103 microseconds
Maximum Execution Time	236 microseconds
Minimum Execution Time	24 microseconds

The routine was timed in a 13.0 Mhz 8096 as it calculated the square roots of all positive 32-bit numbers. The following numbers include the CALL and return sequence and were measured using Timer 1 of the 8096.

ROUTINE FOR THE 8096

08204h or 4634h. The largest square root that we can generate from a 32-bit radicand can be represented in 16-bits. If we are careful in picking our initial Xold we can do all of the divisions with the 32 by 16 divide instruction we have available. Picking the largest possible 16-bit number (0fffh) will always work although it may slow the calculation down by requiring too many iterations to arrive at the correct result. The algorithm below takes a slightly more intelligent approach. It uses the normalize instruction to figure out how many leading zeros the 32-bit radicand has and picks an initial Xold based on this information. If there are 16 or more leading zeros then the radicand is less than 16 bits so an initial Xold of 0fffh is chosen. If the radicand is more than 16 bits then the initial Xold is calculated by shifting the value 0fffh by half as many places as there were leading zeros in the radicand. To give credit where credit is due I first saw this "trick" in the January 1996 issue of Dr. Dobbs's Journal in a letter from Michael Barr of McGill University.

The only significant problem in implementing the square root calculation using this algorithm is that the division of R by Xold could easily be a 32 by 32 divide if R is a 32 bit integer. This is ok if you happen to have a 32 by 32 divide instruction, but most 16-bit machines (including the 8096) only provide a 32 by 16 divide. However, a little bit of creative laziness will allow us to square by using the 32 by 16 bit divide on the 8096.

Practice

Note that in integer arithmetic the remainder of a division is ignored and the square root of a number is floored (i.e. the square root is the largest integer which when multiplied by itself, is less than or equal to the radicand).

$$Xnew = (37\sqrt{8} + 8)\sqrt{2} = 8; Xold = 8 - done!$$

$$Xnew = (37\sqrt{10} + 10)\sqrt{2} = 8; Xold = 8$$

$$Xnew = (37\sqrt{18} + 18)\sqrt{2} = 10; Xold = 10$$

$$Xnew = (37\sqrt{1} + 1)\sqrt{2} = 18; Xold = 18$$

the square root of 32 with an initial guess (Xold) of 1: As an example of how it all works, consider taking ing between two values. This is not the case with this technique can sometimes get you in trouble because If you are dealing with real (floating point) numbers loop on the approximation until Xnew stops changing. until you get an answer you like. A common technique

Xnew is the new approximation

square root

Xold is the current approximation of the

where: R is the radicand

$$Xnew = (R/Xold + Xold)\sqrt{2}; Xold = Xnew$$

by repeating the approximation:

Newton showed that the square root can be calculated

Comments 6-202

Practice 6-202

Theory 6-202

Page 6-201

This Application Brief presents an example of calculating the square root of a 32-bit signed integer.

Theory

Newton showed that the square root can be calculated by repeating the approximation:

$$X_{\text{new}} = (R/X_{\text{old}} + X_{\text{old}})/2; X_{\text{old}} = X_{\text{new}}$$

where: R is the radicand

Xold is the current approximation of the square root

Xnew is the new approximation

until you get an answer you like. A common technique for deciding whether or not you like the answer is to loop on the approximation until Xnew stops changing. If you are dealing with real (floating point) numbers this technique can sometimes get you in trouble because it's possible to hang up in the loop with Xnew alternating between two values. This is not the case with integers. As an example of how it all works, consider taking the square root of 37 with an initial guess (Xold) of 1:

$$X_{\text{new}} = (37/1 + 1)/2 = 19; X_{\text{old}} = 19$$

$$X_{\text{new}} = (37/19 + 19)/2 = 10; X_{\text{old}} = 10$$

$$X_{\text{new}} = (37/10 + 10)/2 = 6; X_{\text{old}} = 6$$

$$X_{\text{new}} = (37/6 + 6)/2 = 6; X_{\text{old}} = 6 - \text{done!}$$

Note that in integer arithmetic the remainder of a division is ignored and the square root of a number is floored (i.e. the square root is the largest integer which, when multiplied by itself, is less than or equal to the radicand).

Practice

The only significant problem in implementing the square root calculation using this algorithm is that the division of R by Xold could easily be a 32 by 32 divide if R is a 32 bit integer. This is ok if you happen to have a 32 by 32 divide instruction, but most 16-bit machines (including the 8096) only provide a 32 by 16 divide. However, a little bit of creative laziness will allow us to squeeze by using the 32 by 16 bit divide on the 8096.

The largest positive integer you can represent with a 32-bit two's complement number is 07fff8fffh, or 2,147,483,647. The square root of this number is 0b504h, or 46,340. The largest square root that we can generate from a 32-bit radicand can be represented in 16-bits. If we are careful in picking our initial Xold we can do all of the divisions with the 32 by 16 divide instruction we have available. Picking the largest possible 16-bit number (0ffffh) will always work although it may slow the calculation down by requiring too many iterations to arrive at the correct result. The algorithm below takes a slightly more intelligent approach. It uses the normalize instruction to figure out how many leading zeros the 32-bit radicand has and picks an initial Xold based on this information. If there are 16 or more leading zeros then the radicand is less than 16 bits so an initial Xold of 0fffh is chosen. If the radicand is more than 16 bits then the initial Xold is calculated by shifting the value 0ffffh by half as many places as there were leading zeros in the radicand. To give credit where credit is due, I first saw this 'trick' in the January 1986 issue of Dr. Dobbs's Journal in a letter from Michael Barr of McGill University.

The routine was timed in a 12.0 Mhz 8096 as it calculated the square roots of all positive 32-bit numbers, the following numbers include the CALL and return sequence and were measured using Timer 1 of the 8096.

Minimum Execution Time: 24 microseconds

Maximum Execution Time: 236 microseconds

Average Execution Time: 102 microseconds

Comments

The program module which follows is part of a collection of routines which perform integer and real arithmetic on a software implemented tagged stack. The top element of the stack is call TOS and is in fixed locations in the register file. Since the square root operation only involves TOS, further details of the stack structure are not shown.


```

MCS-96 MACRO ASSEMBLER  SQRT                                05/12/86 10:44:30 PAGE  1
DOS MCS-96 MACRO ASSEMBLER, V1.1
SOURCE FILE: ROOT2.A96
OBJECT FILE: ROOT2.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSE
ERR LOC OBJECT      LINE      SOURCE STATEMENT
1 ;
2 sqrt module
3 ;
4 ; 32 bit integer square root for the 8096
5 ;
6 public qstk_isqrt      TOP← SQUARE_ROOT(TOP)
7 extrn interr:entry     Integer error routine
8 ;
9 ; id stags for stack integer routines
0019 isqrt_id      equ      19h
11 ;
12 ; error codes
13 ;
0000 14 overflow      equ      00h
0001 15 paramerr     equ      01h
0002 16 invalid_input equ      02h
17
001C 18 oseg at 1ch
19 ;
001C 20 ax: dsw 1
001C 21 al equ ax:byte
001D 22 ah equ (ax+1):byte
001E 23 dx: dsw 1
0020 24 cx: dsw 1
0022 25 bx: dsw 1
0018 26 sp      equ 18h:word
27
28
0030 29 oseg at 30h
30 ;
0030 31 qstk_reg:
0030 32 dsl 1
0030 33 next      equ qstk_reg:word
0032 34 tos_tag equ (qstk_reg+2):word
0034 35 tos_value:
0034 36 dsl 1
37
38 cseg
39 ;
40 bl macro param
41 bnc param
42 endm
43
44 bhe macro param
45 bc param
46 endm
47 $eject

```

```

MCS-96 MACRO ASSEMBLER  SQRT                                05/12/86 10:44:30 PAGE  2
ERR LOC OBJECT      LINE      SOURCE STATEMENT
0000                48      cseg
0000                49      ;
0000                50      ;
0000                51      qstk_isqrt:
0000                52      ; Takes the square root of the long integer in TOS
0000                53      ; TOS→ Top of argument stack
0000                54      ; iTOS - ISQRT(TOS)
0000                55      ;
0020                56      Xold set cx
0000 A0341C          57      ld      ax,tos_value
0003 A0361E          58      ld      dx,(tos_value+2)
0006 371F07          59      bbc      (dx+1),7,qsi05      ; if (TOS < 0)
0009 C90119          60      push    #(isqrt_id*256+paramerr)
000C EF0000          E 61      call    interr      ; Call interr.
000F F0             62      ret      ; Exit
0010             63      qsi05:
0010 0F221C          64      normal ax, bx
0013 DF3B           65      be      qstk_isqrt0
0015 991022          66      cmpb    bx,#16      ; if (TOS < 2**16)
0018 DA06           67      bbe      qsi10
001A 11FF0020        68      ld      Xold, #Offh      ; Use Offh as first estimate.
001E 200A           69      br      qstk_isqrtloop
0020             70      qsi10:
0020 180122          71      shrb    bx,#1      ; else
0023 A1FFFFF20       72      ld      Xold, #Offfffh      ; Base the first estimate on the
0027 082220          73      shr     Xold, bx      ; number of leading zeroes in TOS.
002A             74      qstk_isqrtloop;
002A A0341C          75      ld      ax,tos_value      ; do
002D A0361E          76      ld      dx,(tos_value+2)      ; if (The divide will overflow)
0030 88201E          77      cmp     dx,Xold      ; The loop is done.
0030             78      bhe      qstk_isqrt_done
0035 8C201C          80      divu    ax,Xold      ; if ( (ax=TOS/Xold) >= Xold)
0038 88201C          81      cmp     ax,Xold      ; The loop is done.
0038             82      bhe      qstk_isqrt_done
003D 0122           84      clr     bx      ; Xold=(ax+Xold)/2
003F 641C20          85      add     Xold,ax
0042 A40022          86      addc    bx,0
0045 0C0120          87      shr     Xold,#1
0048 27E0           88      br      qstk_isqrtloop      ; while (The loop is not done)
004A             89      qstk_isqrt_done:
004A A02034          90      ld      tos_value,Xold      ; TOS=00:Xold
004D A00036          91      ld      (tos_value+2),0
0050             92      qstk_isqrt0:
0050 F0             93      ret      ; Exit
0051             94      end

```

ASSEMBLY COMPLETED. NO ERROR(S) FOUND.



APPLICATION NOTE

AP-406

December 1987

MCS[®]-96 Analog Acquisition Primer

DAVID P. RYAN
INTEL CORPORATION

6

Order Number: 270365-001

ANALOG ACQUISITION PRIMER

CONTENTS

PAGE

INTRODUCTION	6-208
WHAT IS AN ANALOG ACQUISITION SYSTEM?	6-209
A/D CONVERTER	6-209
THE MULTIPLEXER	6-213
SAMPLE-AND-HOLD	6-215
THE MCS®-96 CONVERSION SEQUENCE	6-215
APPLICATION HINTS	6-217
ANALOG INPUTS	6-217
ANALOG REFERENCES	6-218
GETTING MORE RESOLUTION	6-219
APPENDIX A: A/D GLOSSARY OF TERMS	6-222
APPENDIX B: CAPACITIVE INTERPOLATION	6-224
APPENDIX C: ERROR FORMULAS	6-230
APPENDIX D: SAMPLE CONVERTER DATA	6-236
APPENDIX E: BIBLIOGRAPHY	6-305

DAVID P. RYAN
INTEL CORPORATION

CONTENTS PAGE

LISTING OF FIGURES

Figure 1. An Analog Acquisition System	6-209
Figure 2. Ideal A/D Characteristic	6-210
Figure 3. A Three-Bit D-to-A	6-211
Figure 4. Actual and Ideal Characteristics	6-212
Figure 5. Types of Linearity Errors	6-212
Figure 6. Undesirable Converter Operation	6-213
Figure 7. Terminal Based Characteristic	6-214
Figure 8. Repeatability Error	6-213
Figure 9. Sample-and-Hold Voltage	6-215
Figure 10. A/D Converter Block Diagram	6-216
Figure 11. Idealized A/D Sampling Circuitry	6-217
Figure 12. Suggested A/D Input Circuit	6-217
Figure 13. (a). Non-Inverting Buffer	6-218
Figure 13. (b). Inverting Buffer	6-218

For any users of an MCS-96 analog acquisition system (experienced or not), this document contains very useful information. It should be considered mandatory reading in addition to the latest Embedded Controller Handbook and MCS-96 data sheet for the actual device in use prior to the actual design.

scated are drawn from the body of application literature publicly available on the components of an analog acquisition system. There is usually no single meaning for a particular term or specification used to describe analog acquisition. However, there is in most cases a generally accepted definition which is most often used. To the extent possible, we have adopted the most widely definition. To avoid any ambiguity, Appendix A has the dictionary of terms as used to refer to the analog acquisition systems of MCS-96 devices.

CONTENTS PAGE

Figure 14. Trimming Offset and Gain	6-218
Figure 15. Supply Decoupling	6-218
Figure 16. A Flexible Input Circuit	6-219
Figure 17. A Low-Cost Log Amplifier	6-220
Figure 18. A Low Pass Filter	6-220
Figure 19. Dither	6-221
Figure 20. Software Controlled Offset and Gain	6-221
Figure B1 (a). Connections During the Sample Window	6-225
Figure B1 (b). Connections After the Sample Window Closes	6-225
Figure B2. Superposition Analysis of Comparator Input Voltage	6-226
Figure B3. Initial Conditions	6-226
Table D1. Sample Converter Data	6-236

LISTINGS

Listing B1 A/D Converter Simulator	6-228
Listing C1 Error Formulas	6-231

chip-sets teamed with analog acquisition chip sets.

There are two obvious avenues for system cost reduction when a 16-bit CPU is teamed with an on-chip analog acquisition system. For example, closed-loop servo control had been implemented almost exclusively by using analog methods. When an MCS-96 device is designed into such an application, it is not only replacing a microcontroller or microprocessor, but it also replaces closed-loop analog circuitry which never before came in contact with the digital system.

To take full advantage of this new level of integration, digital designers must become familiar with analog acquisition, and analog designers must become familiar

THE MCS®-96 ANALOG ACQUISITION PRIMER

INTRODUCTION

As technology advances, embedded control applications continue to reduce chip-count and demand microcontrollers with increased features to assist system-cost reduction. Since every embedded control application interfaces with the physical world, and the physical world is an analog process, it was inevitable that microcontrollers would include integrated analog acquisition capabilities.

The first such integration of standard microcontroller and A/D converter occurred on Intel's 8022 in 1978. This opened the door to cost reduction of high volume applications that required analog inputs. The device fit well into applications that needed processing of analog data. But this chip, with its 8-bit CPU, could not perform in high-end applications requiring analog inputs, or in applications that had computationally demanding analog tasks.

With the introduction of the MCS®-96 family of 16-bit microcontrollers in 1982, the combined CPU and A/D performance became available to greatly reduce the system cost of mid- and high-performance embedded control applications. These are applications which were customarily implemented with 16-bit microprocessor chip-sets teamed with analog acquisition chip sets.

There are less obvious avenues for system cost reduction when a 16-bit CPU is teamed with an on-chip analog acquisition system. For example, closed-loop servo control had been implemented almost exclusively by using analog methods. When an MCS-96 device is designed into such an application, it is not only replacing a microcontroller or microprocessor, but it also replaces closed-loop analog circuitry which never before came in contact with the digital system.

To take full advantage of this new level of integration, digital designers must become familiar with analog acquisition, and analog designers must become familiar

with digital methods of processing analog signals. This Application Note assists with the first task—understanding of an analog acquisition system.

Designers experienced with analog design, or analog acquisition systems, may find no revelations herein. To those unfamiliar with analog acquisition systems, this Ap Note provides a tutorial on the subject and will serve as a handy reference.

Answering the limitless number of analog circuit design questions is beyond the scope of this Ap Note. Suffice it to say that the effort placed on the design of analog circuits should increase with a decreasing error budget.

At a minimum, the applications literature of op-amp manufacturers and analog design manuals are a good place to start. Furthermore, the applications literature of monolithic analog acquisition system manufacturers should be consulted since the suggestions presented therein are largely transportable to any A/D system.

This Ap Note is organized in the following sections. The components of an analog acquisition system and the errors associated with each is first explained. Then, interfacing suggestions and ideas for getting more resolution are presented. Finally, a set of appendices provides back-up information, a bibliography, actual converter data and some program listings.

The definitions of terms used, and the examples presented, are drawn from the body of applications literature publicly available on the components of an analog acquisition system. There is usually no single meaning for a particular term or specification used to describe analog acquisition. However, there is, in most cases, a generally accepted definition which is most often used. To the extent possible, we have adopted the most used definition. To avoid any ambiguity, Appendix A lists the dictionary of terms as used to refer to the analog acquisition systems of MCS-96 devices.

For any users of an MCS-96 analog acquisition system (experienced or not), this document contains very useful information. It should be considered mandatory reading in addition to the latest Embedded Controller Handbook and MCS-96 data sheet for the actual device in use prior to the actual design.

WHAT IS AN ANALOG ACQUISITION SYSTEM?

An analog acquisition system is a collection of individual units which, when logically configured, form a system capable of converting an analog input to a digital value.

The typical components of an Analog Acquisition Unit (Figure 1) include an Analog-to-Digital Converter (A/D), a Sample-and-Hold (S/H) and an Analog Multiplexer (MUX). The A/D converts the infinitely varying analog voltage present on the S/H into a digital representation for use by the digital system. The S/H is required so a "snapshot" of a changing analog input can be stored for conversion by the A/D. The MUX is used to leverage the investment in the A/D by allowing a large number of isolated analog input channels to use the same converter.

The conversion result of an MCS-96 device is a 10-bit ratiometric representation of the input voltage. This produces a stair-stepped transfer function when the output code is plotted versus input voltage. See Figure 2.

The resulting digital codes can be taken as simple ratiometric information, or they can be used to provide information about absolute voltages or relative voltage changes on the inputs. The more demanding the application is on the A/D converter, the more important it is to fully understand the converter's operation. For simple applications, knowing the absolute error of the converter is sufficient. However, controlling a closed loop with analog inputs necessitates a detailed understanding of an A/D converter's operation and errors.

The errors inherent in an analog-to-digital conversion process are many: quantizing error; zero offset; full-

scale error; differential non-linearity; and non-linearity. These are "transfer function" errors related to the A/D converter. In addition, the S/H and MUX may induce channel dissimilarities and sampling error (described later).

Fortunately, one "Absolute Error" specification is available which describes the sum total of all deviations between the actual conversion process and an ideal converter. The various sub-components of error are, however, important in many applications. These error components are described in Appendix A and in the text below where ideal and actual converters are compared.

A/D Converter

There are at least three well-recognized methods for converting an analog voltage to a digital value—flash, dual slope and successive approximation.

Flash A/Ds are the fastest, and most expensive converters for a given accuracy. Flash converters typically resolve bits of the result in parallel to achieve fast conversions. Flash converter speeds are measured in tens-of-nanoseconds.

Dual slope converters are the slowest, but most accurate. Dual slope conversion is rather insensitive to noise on the input, but conversion times are measured in milliseconds.

Successive approximation converters provide a balanced tradeoff between speed and accuracy. Successive approximation conversion times are measured in tens-of-microseconds, and converter implementations are very economical for a given accuracy.

6

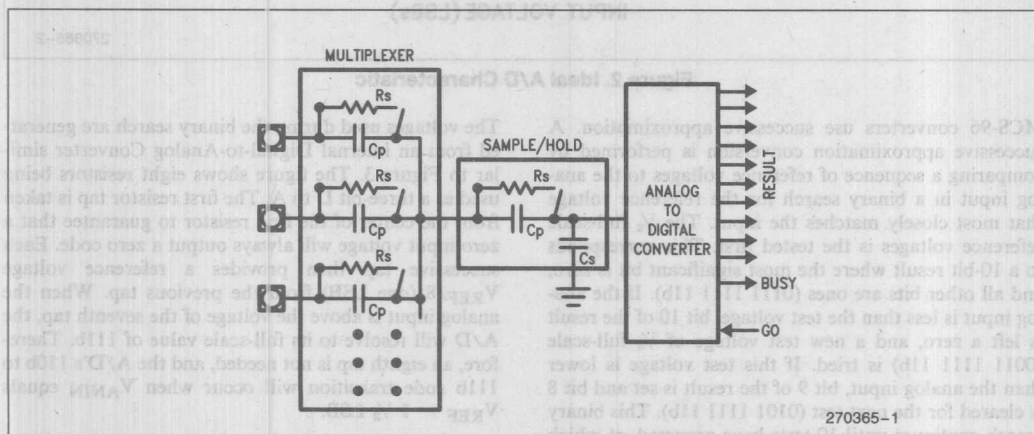


Figure 1. An Analog Acquisition System

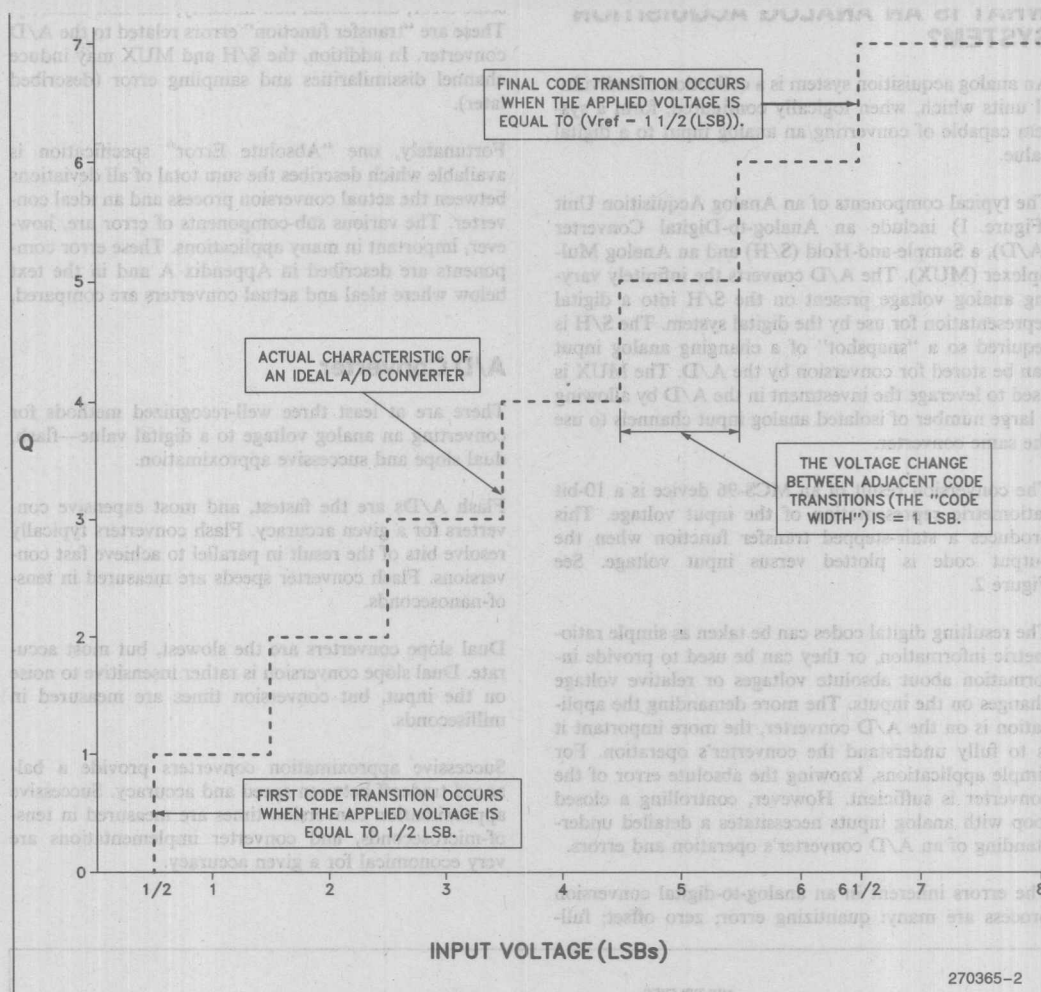


Figure 2. Ideal A/D Characteristic

MCS-96 converters use successive approximation. A successive approximation conversion is performed by comparing a sequence of reference voltages to the analog input in a binary search for the reference voltage that most closely matches the input. The $\frac{1}{2}$ full-scale reference voltages is the tested first. This corresponds to a 10-bit result where the most significant bit is zero, and all other bits are ones (0111 1111 11b). If the analog input is less than the test voltage, bit 10 of the result is left a zero, and a new test voltage of $\frac{1}{4}$ full-scale (0011 1111 11b) is tried. If this test voltage is lower than the analog input, bit 9 of the result is set and bit 8 is cleared for the next test (0101 1111 11b). This binary search continues until 10 tests have occurred, at which time the valid 10-bit conversion result resides in a register where it can be read by software.

The voltages used during the binary search are generated from an internal Digital-to-Analog Converter similar to Figure 3. The figure shows eight resistors being used as a three-bit D to A. The first resistor tap is taken from the center of the first resistor to guarantee that a zero input voltage will always output a zero code. Each successive tap then provides a reference voltage $V_{REF}/8$ (one LSB) from the previous tap. When the analog input is above the voltage of the seventh tap, the A/D will resolve to its full-scale value of 111b. Therefore, an eighth tap is not needed, and the A/D's 110b to 111b code transition will occur when V_{ANIN} equals $V_{REF} - 1\frac{1}{2}$ LSB.

The first error seen in this process is unavoidable, and results from the conversion of a continuous voltage to

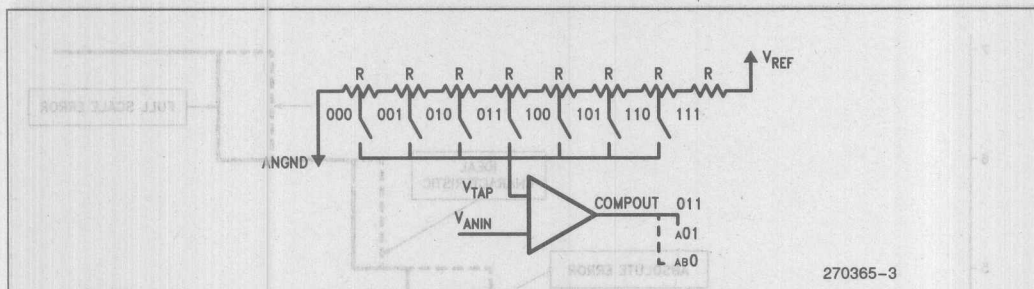


Figure 3. A Three-Bit D-to-A

an integer digital representation. This error is called quantizing error, and is always ± 0.5 LSB. Quantizing error is the only error seen in a perfect A/D converter, and is obviously present in actual converters. Figure 2 shows the transfer function for an ideal 3-bit A/D converter (i.e. the Ideal Characteristic).

Note that in Figure 2 the Ideal Characteristic possesses unique qualities: its first code transition occurs when the input voltage is 0.5 LSB; its full-scale code transition occurs when the input voltage equals the full-scale reference minus 1.5 LSB; and its code widths are all exactly one LSB. These qualities result in a digitization without offset, full-scale or linearity errors. In other words, a perfect conversion.

Figure 4 shows an Actual Characteristic of a hypothetical 3-bit converter which is not perfect. When the Ideal Characteristic is overlaid with the imperfect characteristic, the actual converter is seen to exhibit errors in the location of the first and final code transitions and code widths. The deviation of the first code transition from ideal is called "zero offset". The deviation of the final code transition from ideal is "full-scale error".

The deviation of the code widths from ideal causes two types of errors. Differential Non-Linearity and Non-Linearity. Differential Non-Linearity is a local linearity error measure, whereas Non-Linearity is an overall linearity error measure. For example, Figure 5a shows a transfer function with a large differential non-linearity and a little non-linearity. In contrast, Figure 5b shows a characteristic with small differential errors but a large overall linearity error.

Differential Non-Linearity is the degree to which actual code widths differ from the ideal width. Differential Non-Linearity gives the user a measure of how much the input voltage may have changed in order to produce a one count change in the conversion result.

If the absolute value of an input voltage is less important than the amount that the input changes, the differential non-linearity (DNL) specification of a converter is very important. For example, if the differential non-linearity of a converter is less than ± 0.5 LSB, a one count change in the digital result means that the input voltage changed at most 1.5 LSB (1 LSB ideal ± 0.5 LSB DNL). This is a much more accurate description of the input voltage change than would be available if the differential non-linearity of the converter was not known.

6

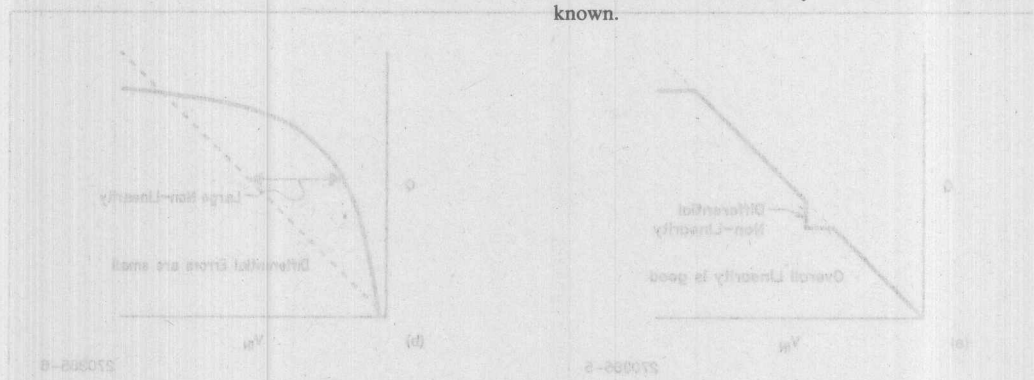
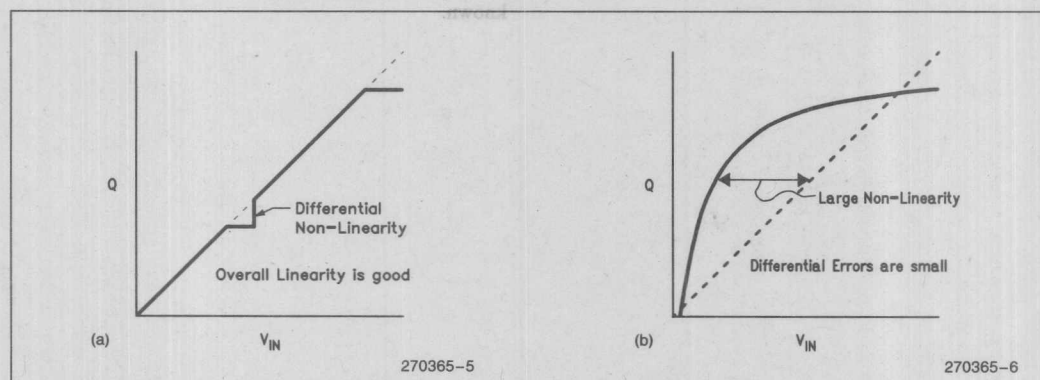
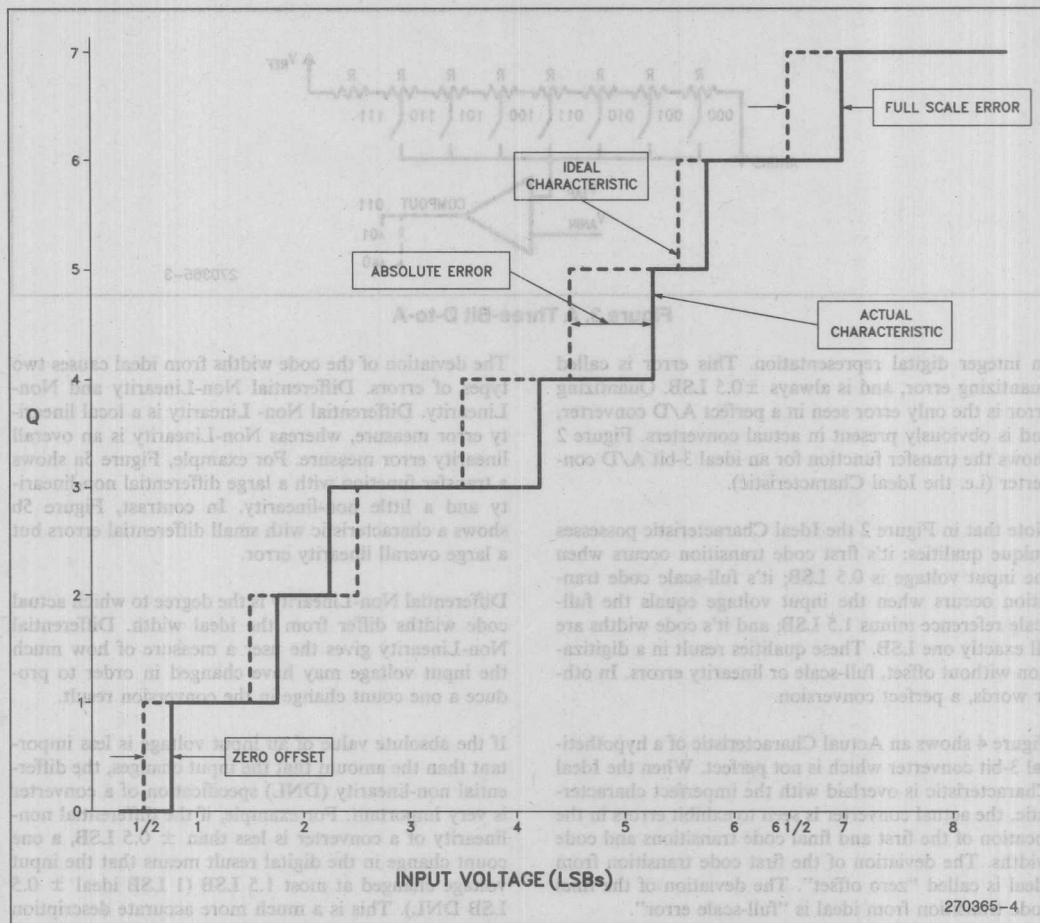


Figure 5. Types of Linearity Errors



Non-Linearity is the worst case deviation of code transitions from the corresponding code transitions of the Ideal Characteristic. Non-Linearity describes how much Differential Non-Linearities could add to produce an overall maximum departure from a linear characteristic.

If the Differential Non-Linearity errors are large enough, it is possible for an A/D converter to miss codes or exhibit non-monotonicity. Neither behavior is desirable in a closed-loop system. A converter has no missed codes if there exists for each output code a unique input voltage range that produces that code only. A converter is monotonic if every subsequent code change represents an input voltage change in the same direction. Figure 6a shows a converter with missed codes. Figure 6b shows a non-monotonic converter.

Differential Non-Linearity and Non-Linearity are quantified by measuring the Terminal Based Linearity Errors. A Terminal Based Characteristic results when an Actual Characteristic is shifted and scaled to eliminate zero offset and full-scale error (see Figure 7). The Terminal Based Characteristic is similar to the Actual Characteristic that would be seen if zero offset and full-scale error were externally trimmed away. In practice, this is done by using input circuits which include gain and offset trimming. (See the Application Hints section for more details.)

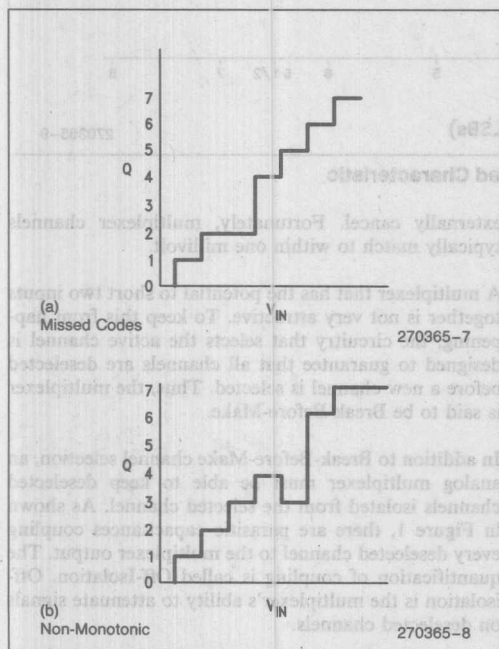


Figure 6. Undesirable Converter Operation

An often overlooked characteristic of A/D converters is that code transitions do not really occur instantaneously at some finite set of input voltages. Specific code transitions can be analyzed by doing repeated conversions around the transition point using a high accuracy input voltage. When this is done, we find that there is actually a range of voltages around code transitions where both the lower and upper codes occur for repeated conversions on the same input voltage.

Figure 8 shows this "repeatability" error. At the lower end of the region of repeatability error the lower code is most prevalent, but the upper code will occur in a small percentage of the conversion attempts. As the input voltage increases slightly, a point is reached where both lower and upper codes occur with 50 percent probability. As the input voltage moves slightly higher, the upper code occurs most often with the lower code showing up in a small percentage of conversions.

The repeatability error is due to the fundamental ability of the comparator in the A/D to resolve very similar voltages. Random noise also contributes to repeatability errors. On MCS-96 devices, the width of the region of repeatability error has been found to be typically 1 mV to 1.25 mV. Since this error is specified, all other errors are specified assuming the code transitions occur at the voltage where adjacent codes are equally likely.

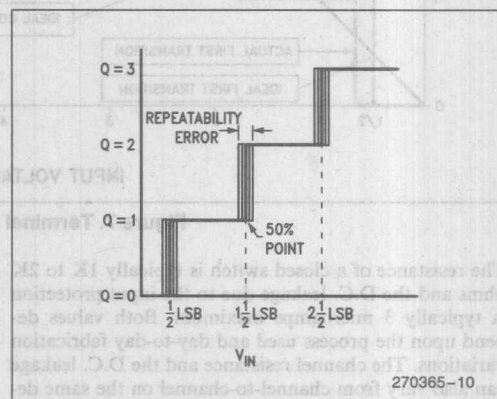


Figure 8. Repeatability Error

The Multiplexer

The eight channel multiplexer is implemented as a collection of eight MOS switches. Only one of eight can be closed at any instant in time. Figure 1 shows the multiplexer with the switches acting as resistors when closed and as small parasitic capacitors when open. The input protection devices on the analog input pins are also considered a part of the multiplexer.

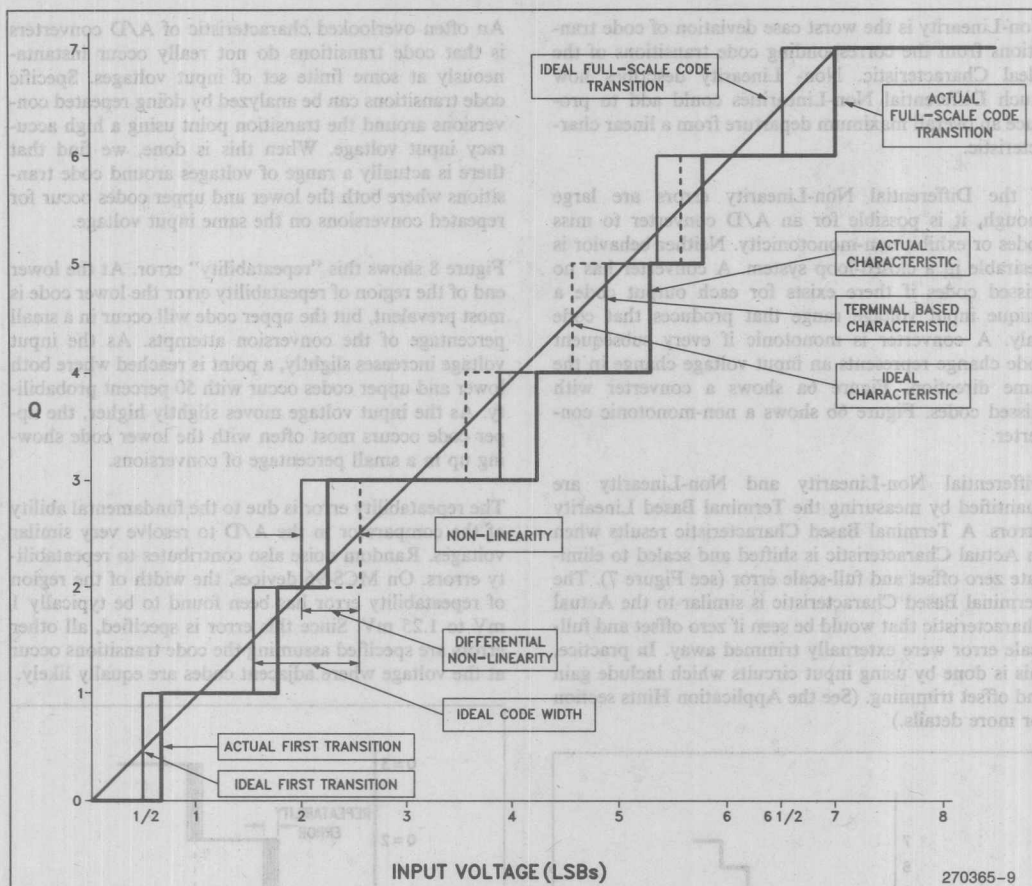


Figure 7. Terminal Based Characteristic

The resistance of a closed switch is typically 1K to 2K ohms and the D.C. leakage due to the input protection is typically 3 microamps maximum. Both values depend upon the process used and day-to-day fabrication variations. The channel resistance and the D.C. leakage can also vary from channel-to-channel on the same device. These variations can be seen in the conversion process and are described by the channel-to-channel matching specification.

Channel-to-channel matching specifies the input voltage differences induced by mismatched elements of the multiplexer. This error is quantified by measuring the difference between the input voltages necessary to cause the same code transition to occur through different multiplexer channels under identical test conditions.

Matching errors are more complex than a simple voltage offset between channels, and thus are difficult to

externally cancel. Fortunately, multiplexer channels typically match to within one millivolt.

A multiplexer that has the potential to short two inputs together is not very attractive. To keep this from happening, the circuitry that selects the active channel is designed to guarantee that all channels are deselected before a new channel is selected. Thus, the multiplexer is said to be Break-Before-Make.

In addition to Break-Before-Make channel selection, an analog multiplexer must be able to keep deselected channels isolated from the selected channel. As shown in Figure 1, there are parasitic capacitances coupling every deselected channel to the multiplexer output. The quantification of coupling is called Off-Isolation. Off-isolation is the multiplexer's ability to attenuate signals on deselected channels.

Sample-and-Hold

The sample-and-hold of an analog acquisition system can be built using an analog switch and a sample capacitor. As with the multiplexer, there is also a parasitic capacitance coupling the switch input to the sample capacitor when the switch is open (Figure 1).

The resistance of the sample-and-hold switch combines with the series resistance of the multiplexer to impede the current necessary to charge the sample capacitor. For example, with a 5K ohm total input resistance from the pin to the 2 pf sample capacitor, the RC time constant is 10 nS ($2 \text{ pf} \times 5 \text{ K ohms}$).

During the one microsecond that the sample capacitor is connected to the input, 100 time constants elapse (1 microsecond/10 nS). This means that the sample capacitor is 100 percent of the voltage on the input pin ($1 - e^{-100}$), assuming a zero source impedance.

If a source impedance of 2K ohms is assumed, the RC time constant of the sampling process would be 14nS ($7 \text{ K ohms} \times 2 \text{ pf}$). Thus, 71.4 time constants would pass in one microsecond resulting in the sample capacitor being charged to within 99.9 percent of its final value. Source impedances above 2K ohms would begin to degrade the conversion accuracy due to D.C. leakage (described later).

Figure 9 shows the actual input voltage and the sampled voltage approaching the input voltage. Once the sample-and-hold switch closes, the sample window begins. The sample window extends for four state times and ends with the sample-and-hold switch opening on MCS-96 devices (except 8X9X-90, which is 8 state times and has no sample-hold). Figure 9 also shows the sample delay, which is the delay from the time a start conversion signal is generated to the time a conversion process begins.

It is important to understand the uncertainties associated with the timing of the sample-and-hold. Digital signal processing algorithms rely upon the "spectral purity" of the sampling process. If the sample window jumps around with respect to the start conversion signal, or if the start conversion signal cannot be generated at precise times, consecutive samples of input data will not be equally spaced in time (i.e. sampling will be spectrally impure).

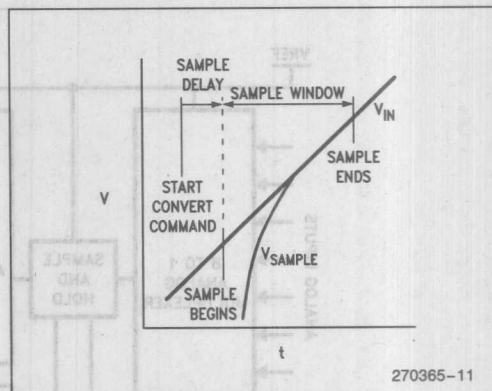


Figure 9. Sample-and-Hold Voltage

To improve the spectral purity of the sampling in digital signal processing applications, sequential MCS-96 start conversion signals can be generated with less than 50 nanoseconds of jitter using the HSO unit. The sample delay and sample time are also a constant number of state times to within 50 nanoseconds each.

Once the sample window closes, it is desired that all further changes on any input channel be isolated from the sample capacitor. The multiplexer's off-isolation is responsible for isolating deselected channels, while the sample-and-hold switch must attenuate changes on the selected channel. This source of error is described as Feedthrough. Feedthrough is quantified as the ability of the sample-and-hold to reject unwanted signals on its input.

Other factors that affect a real A/D Converter system include sensitivity to temperature. Temperature sensitivities are described by the change in typical specifications with a change in temperature.

The MCS®-96 Conversion Sequence

The MCS-96 Analog Acquisition System includes an eight channel analog multiplexer, sample-and-hold circuit and 10-bit analog to digital converter (Figure 10). An MCS-96 device can therefore select one of eight analog inputs, sample-and-hold the input voltage and convert the voltage into a digital value. Each conversion takes 22 microseconds (8097BH), including the time required for the sample-hold (with XTAL1 = 12 MHz). The method of conversion is successive approximation:

For the sample delay, the multiplexer output is connected to the sample capacitor and remains connected for four state times (sample time). After this four state time "sample window" closes, the input to the sample capacitor is disconnected from the multiplexer so that changes on the input pin will not alter the stored charge while

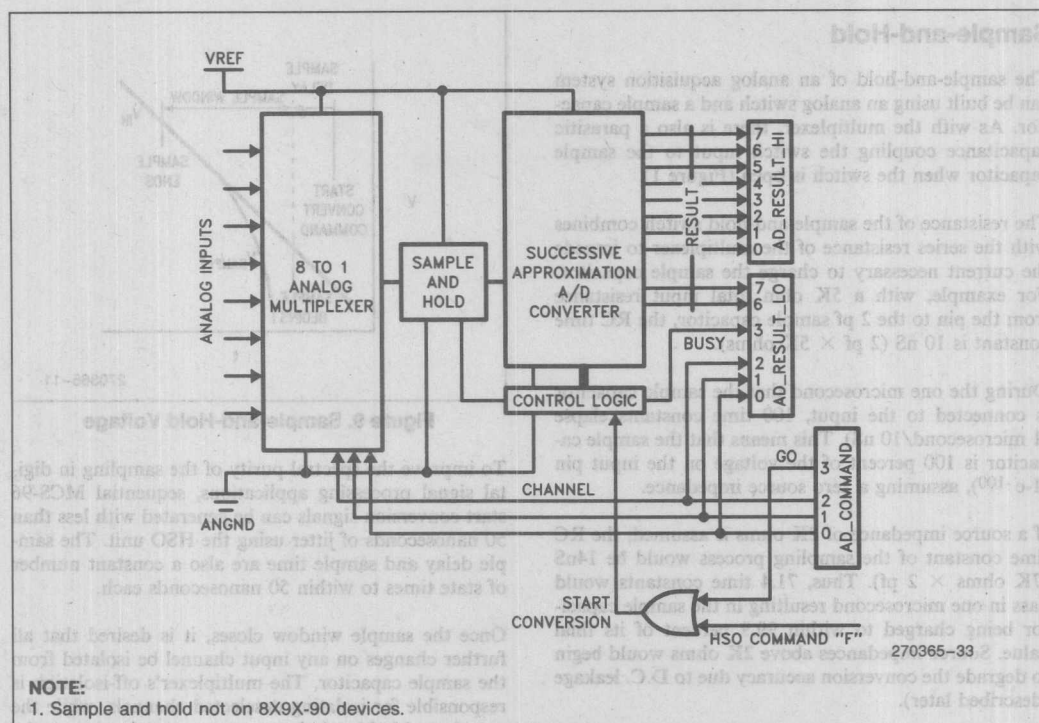


Figure 10. A/D Converter Block Diagram

The conversion process is initiated by the execution of HSO command OFH, or by writing a one to the GO Bit in the A/D Control Register. Either activity causes a start conversion signal to be sent to A/D control logic. If an HSO command was used, the conversion process will begin when Timer 1 increments. This aids applications attempting to approach spectrally pure sampling, since successive samples spaced by equal Timer 1 delays will occur with a variance of about ± 50 ns (assuming a stable clock on XTAL1). However, conversion initiated by writing a one to the ADCON register GO Bit will start within three state times after the instruction has completed execution, resulting in a variance of about $0.75 \mu\text{s}$ ($\text{XTAL1} = 12 \text{ MHz}$).

Once the A/D unit receives a start conversion signal, there is a one state time delay before sampling (sample delay) while the successive approximation register is reset and the proper multiplexer channel is selected. After the sample delay, the multiplexer output is connected to the sample capacitor and remains connected for four state times (sample time). After this four state time "sample window" closes, the input to the sample capacitor is disconnected from the multiplexer so that changes on the input pin will not alter the stored charge while

the conversion is in progress. The sample delay and sample time uncertainties are each approximately ± 50 ns, independent of clock speed.

To perform the actual analog-to-digital conversion the MCS-96 implements a successive approximation algorithm. The converter hardware consists of a 256-resistor ladder, a comparator, coupling capacitors and a 10-bit successive approximation register (SAR) with logic that guides the process. The resistor ladder provides 20 mV steps ($V_{REF} = 5.12V$), while capacitive coupling is used to create 5 mV steps within the 20 mV ladder voltages. Therefore, 1024 internal reference voltages are available for comparison against the analog input to generate a 10-bit conversion result. Appendix B contains a detailed description of the method used to generate 1024 voltages from a 256-resistor chain.

The total number of state times required for a 10-bit conversion varies from one MCS-96 version to the next. Attempting to short-cycle the 10-bit conversion process by reading A/D results before the done bit is set may work on some versions of MCS-96 devices, however it is not recommended. Short-cycling is not tested, nor is it guaranteed. Furthermore, it may not work on future MCS-96 devices.

APPLICATION HINTS

The analog signals that must be converted by an analog acquisition system vary widely. The analog input may arrive at the controller as a voltage or current. The range may be 0 to 1 volt or ± 30 volts, or some other arbitrary range. The input may be linear, logarithmic, non-linear, or perturbed in some bizarre fashion. Although interfacing to such signals could be considered an art form, some simple suggestions are contained in this section.

Analog Inputs

The external interface circuitry to an analog input is highly dependent upon the application, and can impact converter characteristics. In the external circuit's design, important factors such as input pin leakage, sample capacitor size and multiplexer series resistance from the input pin to the sample capacitor must be considered.

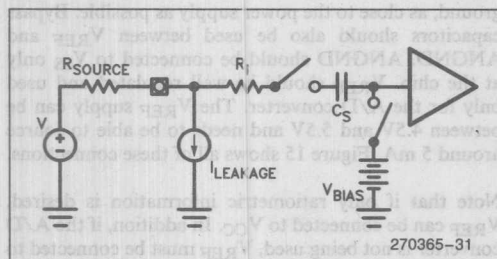


Figure 11. Idealized A/D Sampling Circuitry

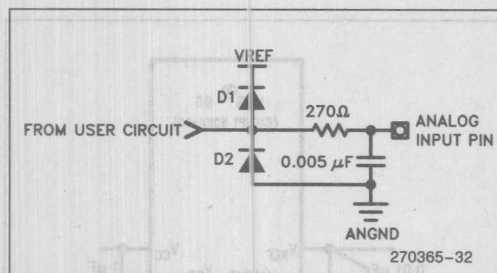


Figure 12. Suggested A/D Input Circuit

For the 8096BH, these factors are idealized in Figure 11. The external input circuit must be able to charge a sample capacitor (C_S) through a series of resistance (R_I) to an accurate voltage given a D.C. leakage (I_L). On the 8096BH, C_S is around 2 pF, R_I is around 5 K Ω and I_L is specified at 3 μ A maximum. In determining the source impedance R_S , V_{BIAS} is not important.

External circuits with source impedances of 1 K Ω or less will be able to maintain an input voltage within a

tolerance of about ± 0.61 LSB ($1.0 \text{ K}\Omega \times 3.0 \mu\text{A} = 3.0 \text{ mV}$) given the D.C. leakage. Source impedances above 2 K Ω can result in an external error of at least one LSB due to the voltage drop caused by the 3 μ A leakage. In addition, source impedances above 25 K Ω may degrade converter accuracy as a result of the internal sample capacitor not being fully charged during the 1 μ s (12 MHz clock) sample window.

Placing an external capacitor on each analog input will reduce the sensitivity to noise, as the capacitor combines with source resistance in the external circuit to form a low-pass filter. In practice, one should include a small series resistance prior to an external low leakage capacitor on the analog input pin and choose the largest capacitor value practical, given the frequency of the signal being converted. This provides a low-pass filter on the input, while the resistor will also limit input current during over-voltage conditions.

Figure 12 shows a simple analog interface circuit based upon the discussion above. The circuit in the figure also provides limited protection against over-voltage conditions on the analog input (limits to 2.6 mA with 270 Ω (0.7/270)). The circuit induces leakage from the diodes, which should be kept small.

The wide range of possible analog environments that must be interfaced to, or the existence of stringent accuracy requirements, makes the consideration of alternative input buffer configurations necessary. The most popular input buffer is a single op-amp in the non-inverting or inverting configurations of Figure 13.

In the non-inverting circuit of Figure 13 (a), the analog input is scaled by the buffer gain to output 5 volts when the input is at its maximum positive input. When the buffer input is 0 volts, the output will also be 0 volts.

In the inverting circuit of Figure 13 (b), a reference equal to the maximum possible input voltage is placed on the non-inverting input of the op-amp and the actual analog input is placed on the inverting input. The output voltage of the buffer is then proportional to the deviation of analog input from its maximum possible value. For example, when the analog input equals V_{MAX} , the buffer output will equal 0 zero volts. When the analog input equals its minimum value, the buffer output equals 5 volts. The digital result from the A/D converter might, of course, have to be complemented before being used.

The circuits of Figure 13 show only feedback resistors that set the gain of the buffer. In practice, it will often be necessary to include offset adjustments, gain trimming, temperature or frequency stability compensation, or components to build an active filter.

Figure 14 depicts a generalized non-inverting input buffer that offsets the analog input and scales the input

to a 5 volt range. The course offset is set by the ratio of R_{BIG1} and R_{BIG2} , while offset fine tuning is done by adjusting R_{TRIM} . The course gain is set by the ratio of R_{G1} and R_{G2} while gain trimming is done with R_{GTRIM} .

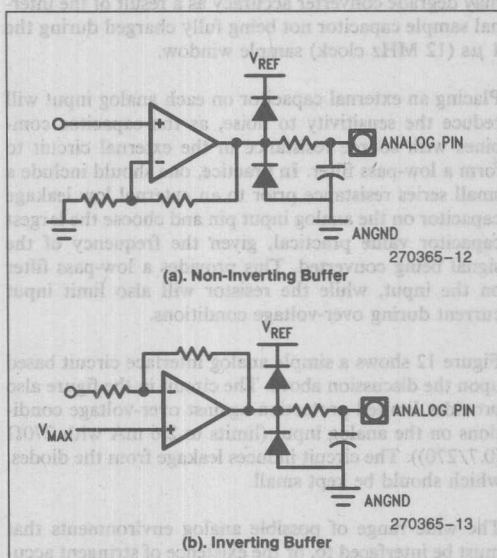


Figure 13

By trimming the offset and gain, not only can external component errors be zeroed out, but the offset and full scale error of the A/D converter can be nulled.

The procedure for nulling offset and gain is simple. First, a voltage is applied to V_{IN} which corresponds to the ideal first code transition of the A/D. R_{TRIM} is adjusted so that 50 percent of the conversion results are 0 while 50 percent are 1. Second, a voltage is applied to V_{IN} which corresponds to the ideal final code transition of the A/D converter. R_{GTRIM} is then adjusted until 50 percent of the conversion results are 3FEH and 50 percent are 3FFH. Once this adjustment is complete, the converter zero offset and full-scale errors are nulled, and could be ignored (except for temperature variation). This allows the system to rely upon the tighter, more descriptive converter specifications for Terminal Based Non-Linearity and Differential Non-Linearity.

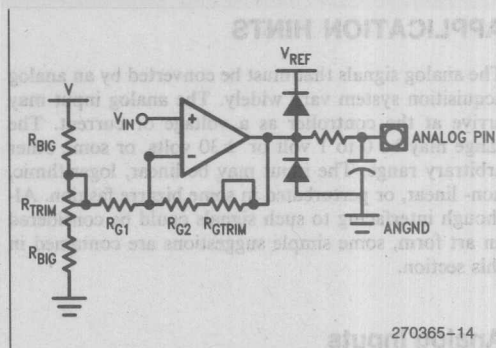


Figure 14. Trimming Offset and Gain

Analog References

Reference supply levels strongly influence the absolute accuracy of the conversion. For this reason, it is recommended that the ANGND pin be tied to a clean ground, as close to the power supply as possible. Bypass capacitors should also be used between V_{REF} and ANGND. ANGND should be connected to V_{SS} only at the chip. V_{REF} should be well regulated and used only for the A/D converter. The V_{REF} supply can be between 4.5V and 5.5V and needs to be able to source around 5 mA. Figure 15 shows all of these connections.

Note that if only ratiometric information is desired, V_{REF} can be connected to V_{CC} . In addition, if the A/D converter is not being used, V_{REF} must be connected to V_{CC} and ANGND to V_{SS} for Port0 to work as a digital port.

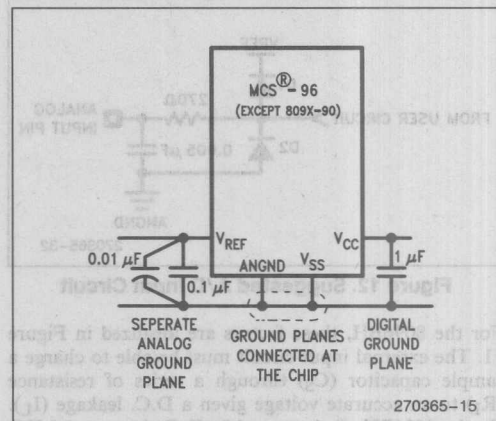


Figure 15. Supply Decoupling

Getting More Resolution

Given that the A/D converter can convert an analog input ranging from 0 volts to 5 volts into 1024 steps of 5 millivolts each, the desire for more resolution can come from three basic needs – need extra LSB, need extra MSB, need BOTH.

The configuration shown in Figure 16 can be used to solve each of the “more resolution” problems. This set-up requires the use of two input channels with different offsets and gains.

When the 5 millivolt step size of the A/D is too large for the application requirements, but the 5 volt range is sufficient, the system needs an “extra LSB”. For example, an application requiring 2.5 millivolt steps over a 5 volt range needs an 11-bit conversion result. The 11th bit needs to be added to the least significant side of the 10-bit result (the “right”). This can be achieved using the circuit of Figure 16.

If both channels are set for a gain of 2, with channel 1 offset to 2.5 volts, the 5 volt input range is split into 2.5 volt ranges that are amplified by two before being input to the A/D. While V_{IN} is between 0 and 2.5 volts, channel 0 will be providing a proportional voltage between 0 volts and 5 volts to the A/D converter and channel 1 will be clamped to 5 volts. When V_{IN} rises above 2.5 volts, channel 1 will begin to output a proportional voltage between 0 volts and 5 volts to the A/D converter and channel 0 will be clamped at 5 volts. Using this method, an 11-bit (2048 step) result is created with 2.5 millivolt steps (i.e. an extra LSB).

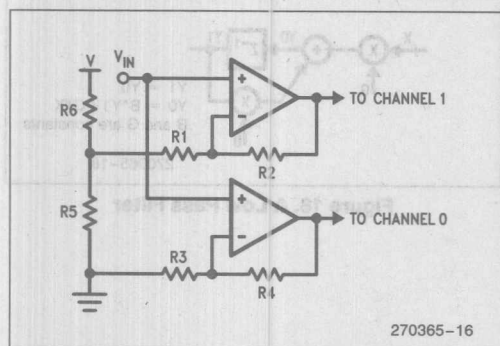


Figure 16. A Flexible Input Circuit

It is useful to note that only one conversion per sample will be required if the software keeps track of which channel is active. The only time that two conversions will be required for one sample is when the voltage crosses the midpoint.

The second reason that “more resolution” is requested is the need for an “extra MSB”. When the converter’s input voltage range is too small (5 volts when 10 volts is needed), but 5 millivolt steps over the actual input voltage range is sufficient, an extra bit is needed on the most significant (“left”) side of the 10-bit result. The circuit of Figure 16 can also be used, with different gains and offsets, to satisfy this extra MSB need by splitting the 10 volt range into 5 volt ranges.

If both channels of Figure 16 are set for unity gain, and channel 1 is offset to 5 volts, an 11-bit conversion result with 5 millivolt steps is available. While V_{IN} is in the lower half of its range (0 volts to 5 volts), channel 0 will be active. While V_{IN} is in the upper half of its range (5 volts to 10 volts), channel 1 will be active. Thus, an extra MSB is created.

For applications requiring multiple extra bits of result, the solutions can become more “elegant” (i.e. elaborate). However, it is profitable to first squeeze the most out of the now familiar circuit in Figure 16.

Assume that the analog input, V_{IN} , ranges from 0 volts to 10 volts, and it is desired to measure this range in 2.5 millivolt steps. This requires two extra bits of result – one extra MSB and one extra LSB. A simple extrapolation of the preceding discussion of creating extra bits might have the designer planning to tieup four channels of the multiplexer needlessly. Needlessly, that is, if the application is a typical control application where the high accuracy requirements are only important in the “normal” operating range of the process. Outside of the normal operating range is the “possible” operating range which must be measured, but with less stringent requirements.

Since the requirements of the normal range set the necessary LSB weight, and the extent of the possible range sets the maximum voltage span, it follows that only two channels need to be used (Figure 16). Channel 0 would be set with a gain that compressed the possible V_{IN} range to 5 volts, while channel 1 would be offset to the normal operating range and would have a gain of two to expand this region of critical interest. With this ap-

proach, 100 percent of the normal operating range is digitized in 2.5 millivolt steps, while 100 percent of the possible range is digitized in 10 millivolt steps.

Unfortunately, not all high resolution applications can be described as a process with a small region of in-control operation, where the process is out-of-control outside of that small region. For example, it is necessary to measure airflow in an engine controlling carburetion. The air flow at idle is likely to be several orders-of-magnitude lower than the airflow at full RPM. The process needs to be in tight control over the entire range, not only when the engine is at half-speed.

When it is desired to measure a process with a fixed percent of error throughout a range spanning several orders-of-magnitude, a non-linear input buffer becomes attractive. For example, assume that the analog signal that needs to be digitized can vary from 1 millivolt to 25 volts and describes a physical process that must be represented digitally with 1 percent error at any point in the possible input range. A linear solution to this application would require a converter with a 10 microvolt LSB ($1\% \times 1 \text{ mV}$), and a resolution of 22 bits ($25 \text{ V}/10 \text{ microvolts}$). This is clearly undesirable.

The use of a log input buffer to compress the 25 volt range logarithmically to 5 volts would satisfy the application requirements. The input would range from 1 millivolt to 25 volts with the output ranging from 0 volts to 5 volts proportionally to the log of $V_{IN}/1\text{mV}$. Each one-percent change in the input voltage would change the output voltage by 5 millivolts (one count). The antilog could be taken in software using a lookup table, or the control calculations could be performed in a log base.

Simple inexpensive log-amps can be built as in Figure 17, or high-accuracy, self-contained log-amps can be purchased. Which is chosen depends upon the application tradeoffs of price and performance.

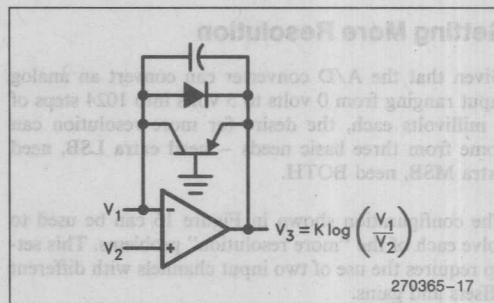


Figure 17. A Low-Cost Log Amplifier

Other techniques become available for consideration in systems that have slow sample rate requirements, but very high resolution requirements. In addition to the methods described above, which require external hardware, software filtering or other post-processing of the conversion results can be productive. Each method relies upon the ability to sample the analog input much faster than the system requires an analog input.

When resolution is limited by filterable noise, perhaps the most straightforward approach to post-processing is to oversample the input by a factor of N and digitally low-pass filter the data (i.e. weighted rolling average). A result would be reported to the rest of the system every N samples (Figure 18). A low-pass filter can increase the signal-to-noise ratio (SNR) by a factor of N (see bibliography). However, care must be taken to be certain that the input voltage varies slowly with respect to the sampling rate.

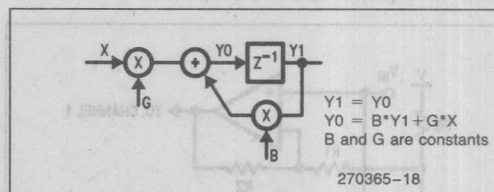


Figure 18. A Low Pass Filter

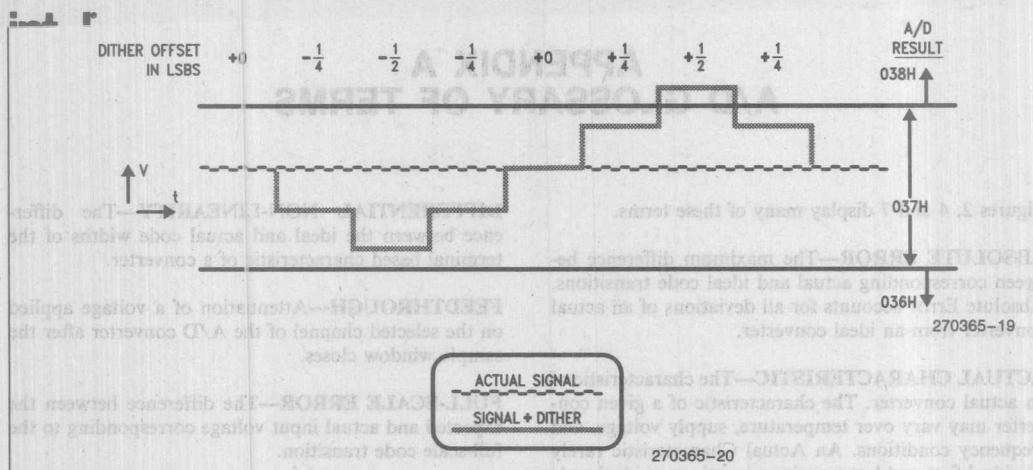


Figure 19. Dither

Another approach to creating more resolution is called "synchronized dither". Figure 19 shows an input voltage that is constant somewhere between two code transition points. This input is "dithered" by adding a small periodic waveform ($1/4$ LSB steps) to the input while performing an A/D conversion synchronized with each dither step. Every time the dither completes a full cycle, the eight conversion results are averaged to form one digitized value. Since the dither is periodic and symmetrical about 0 volts, its average impact on the input voltage is 0 volts.

The creation of extra resolution can be seen with the example shown in Figure 19. Without dither, the input voltage would always convert to 37H. With dither, one-eighth of the conversions would be 38H and $7/8$ of the conversions would be 37H. If every eight conversions were averaged, the result would be $37H + 1/8$ LSB. The possible results given a four level dither, where the input voltage was always within the 37H code width, would be:

$$\begin{aligned} 36H + 5/8 \\ 36H + 7/8 \\ 37H + 0 \\ 37H + 1/8 \\ 37H + 3/8 \end{aligned}$$

Hence, four new levels exist (two bits).

Dither will only create more resolution up to the limit of the A/D converter comparator's ability to distinguish voltages. Since MCS-96 converter repeatability error is typically around 1 millivolt to 1.25 millivolts, $1/4$ LSB dither is the practical limit if no other processing is done. Figure 20 shows a simple method by which

the input voltage could be dithered under software control.

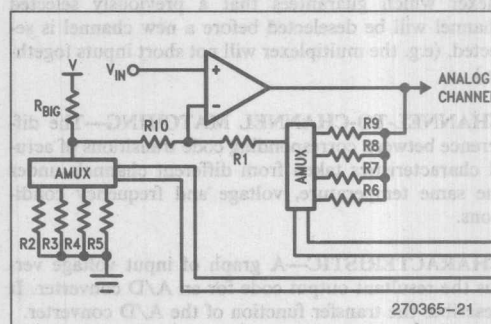


Figure 20. Software Controlled Offset and Gain

While only a few of the more obvious interfacing techniques were described here, there are as many innovative interfacing tricks as there are designers.

CONCLUSION

This application note provides a fundamental understanding of MCS-96 analog acquisition for the digital designer. Since answering the limitless number of analog circuit design questions is beyond the scope of this document, it is expected that analog design manuals and the large body of publicly available applications literature will be consulted for detailed design hints. Furthermore, the applications literature of monolithic analog acquisition system manufacturers should be consulted since the suggestions presented therein are largely transportable to any A/D system.

APPENDIX A A/D GLOSSARY OF TERMS

Figures 2, 4 and 7 display many of these terms.

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An Actual Characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversions under the same conditions.

BREAK-BEFORE-MAKE—The property of a multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g. the multiplexer will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE CENTER—The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION—The point at which the converter changes from an output code of Q , to a code of $Q + 1$. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK—See "Off-Isolation".

D.C. INPUT LEAKAGE—D.C. Leakage current of an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic of a converter.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D converter after the sample window closes.

FULL-SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full-scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at $V_{IN} = 0.5 \text{ LSB}$, its last code transition at $V_{IN} = (V_{REF} - 1.5 \text{ LSB})$ and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB - LEAST SIGNIFICANT BIT—The voltage value corresponding to the full-scale voltage divided by 2^n , where n is the number of bits of resolution of the converter. For a 10-bit converter with a reference voltage of 5.12 volts, one LSB is 5.0 mV. Note that this is different than digital LSBs, since an uncertainty of two LSBs, when referring to an A/D converter, equals 10 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 20 mV.)

MONOTONIC—The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES—For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY—The maximum deviation of code transitions of the terminal based characteristic from the corresponding code transitions of the actual characteristic of a converter.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY—The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY—The variation in the Sample Delay.

SAMPLE TIME—The time that the sample window is open.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

During the sample window (Figure B1b), V_{ANIN} and V_{OPG} control the amount of charge stored in C_A and C_B (V_{OPG} controls the converter offset). Once the sample window closes (Figure B1b), voltages applied to V_{IN} and V_{IN} will add or subtract charge proportional to $(V_{\text{ANIN}} - V_{\text{IN}})$ on C_A and $(V_{\text{OPG}} - V_{\text{IN}})$ on C_B . The inverting comparator input of Figure B1b will remain at V_{BIAS} due to the charges on C_A and C_B . The non-inverting comparator input will always remain at V_{BIAS} and serves as a reference.

If a V_{IN} combination is applied which causes the non-inverting input to drop below V_{BIAS} the comparator will output a 1 to indicate that the applied voltage was lower than the original V_{ANIN} . To better understand how the circuit works, Figure B2 shows the superposition analysis used to form the equation for V_{OUT} given initial charge on C_A and C_B and new input voltages V_{IN} and V_{IN} .

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION—An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An Actual Characteristic which has been rotated and translated to remove zero offset and full-scale error.

V_{CC} REJECTION—Attenuation of noise on the V_{CC} line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

1024 resistor chain.

Before beginning a detailed description of the capacitive part of the conversion process, it is necessary to understand a few details about the resistor chain.

There are 526 resistors connected in series from the analog reference to analog ground. The actual value of the resistors only impacts the current through the resistor chain. If every resistor in the chain is the same value the converter will function properly.

To reduce resistor-to-resistor variation, the chain is folded in half, and then in an accordion fashion to produce a 16×16 block of resistors. This minimizes the sensitivity of the array to processing gradients, while also allowing the array to be addressed roughly similar to a 16×16 memory array.

APPENDIX B CAPACITIVE INTERPOLATION

A successive approximation A/D converter needs an internal D/A converter of the same resolution as the desired A/D result. A 10-bit D/A could have been made using a string of 1024 resistors connected from the analog reference at one end to ground at the other end. Although this would be technically ideal, such a circuit would be enormous. Therefore, a method was developed to generate the needed reference voltages using a small area of silicon so that an on-chip 10-bit A/D converter would be economical.

The method used relies upon a 256-resistor chain to generate reference voltages in 20mV (5.12V/256) steps while two ratioed capacitors are used to capacitively "interpolate" voltages in-between the resistor tap voltages. The area of the 256-resistor chain together with the capacitors is one-fourth the area of the would-be 1024 resistor chain.

Before beginning a detailed description of the capacitive part of the conversion process, it is necessary to understand a few details about the resistor chain.

There are 256 resistors connected in series from the analog reference to analog ground. The actual value of the resistors only impacts the current through the reference pin. If every resistor in the chain is the same value the converter will function properly.

To reduce resistor-to-resistor variation, the chain is folded in half, and then in an accordion fashion to produce a 16×16 block of resistors. This minimizes the sensitivity of the array to processing gradients, while also allowing the array to be addressed roughly similar to a 16×16 memory array.

As explained earlier, it is desired for the A/D converter to have its first code transition at $\frac{1}{2}$ LSB followed by subsequent code widths 1 LSB wide.

To accomplish this, each resistor is tapped in its center rather than between resistors. For example, the first resistor tap is half-way up the first resistor. This means that the zero resistor tap will output 10mV (20mV/2). When calculating the voltage on a certain resistor tap, you must add 10mV to the product of the tap number and 20mV.

The internal connections while an analog input is being sampled are shown in Figure B1a. Once sampling is complete, the analog input is disconnected and the comparator inputs are no longer clamped to V_{BIAS} (Figure B1b).

During the sample window (Figure B1a), V_{ANIN} and V_{OFS} control the amount of charge stored in C_A and C_B (V_{OFS} controls the converter offset). Once the sample window closes (Figure B1b), voltages applied to V_{IN} and V_{IN2} will add or subtract charge proportional to $(V_{ANIN} - V_{IN})$ on C_A and $(V_{OFS} - V_{IN2})$ on C_B . Unless a voltage is applied to V_{IN} and V_{IN2} . The inverting comparator input of Figure B1b will remain at V_{BIAS} due to the charges on C_A and C_B . The non-inverting comparator input will always remain at V_{BIAS} and serves as a reference.

If a V_{IN} , V_{IN2} combination is applied which causes the non-inverting input to drop below V_{BIAS} the comparator will output a 1 to indicate that the applied voltage was lower than the original V_{ANIN} . To better understand how the circuit works, Figure B2 shows the superposition analysis used to form the equation for V_{OUT} , given initial charge on C_A and C_B and new input voltages V_{IN} and V_{IN2} .

Adding the independent effects shown in Figure B2 we have:

$$V_{OUT} = V_1 + V_2 + V_3 + V_4$$

$$V_{OUT} = V_{IN} \left(\frac{C_A}{C_A + C_B} \right) + V_{IN2} \left(\frac{C_B}{C_A + C_B} \right) + V_{AI} \left(\frac{C_A}{C_A + C_B} \right) + V_{BI} \left(\frac{C_B}{C_A + C_B} \right)$$

$$V_{OUT} = (V_{IN} + V_{AI}) \frac{C_A}{C_A + C_B} + (V_{IN2} + V_{BI}) \frac{C_B}{C_A + C_B} \quad (I)$$

The initial conditions on C_A and C_B are set-up as shown in Figure B3.

We can see that:

$$V_{AI} = V_{BIAS} - V_{ANIN} \quad (II)$$

$$V_{BI} = V_{BIAS} - V_{OFS} \quad (III)$$

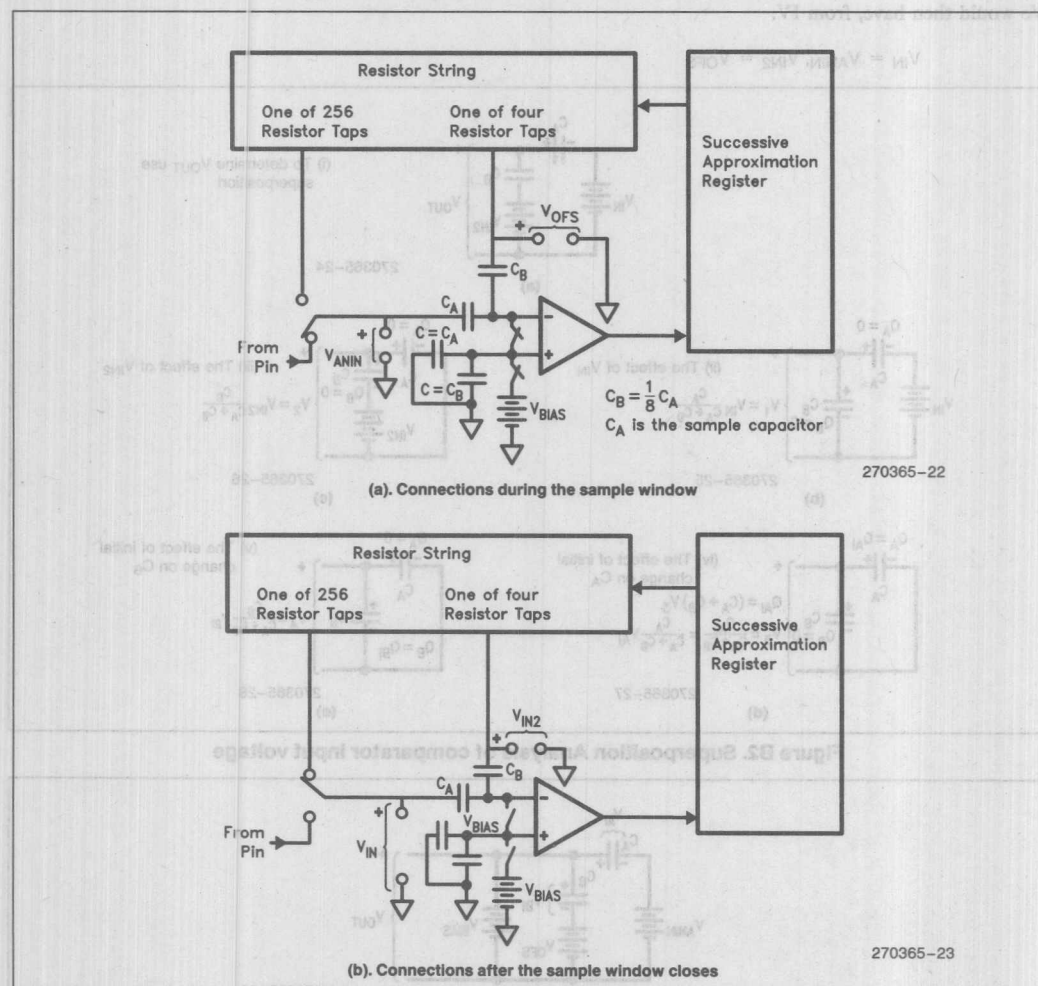


Figure B1

Substituting II and III into I we get:

$$V_{OUT} = (V_{IN} + V_{BIAS} - V_{ANIN}) \frac{C_A}{C_A + C_B} + (V_{IN2} + V_{BIAS} - V_{OFS}) \frac{C_B}{C_A + C_B} \quad (IV)$$

V_{OUT} becomes the input voltage to the comparator which ideally presents no load. The only way to make V_{OUT} approach the value of V_{BIAS} (after V_{BIAS} is removed) is to apply a voltage combination which makes equation IV evaluate to V_{BIAS} . If we had an infinitely variable internal voltage reference to use, we could just set the reference on V_{IN} to the value of V_{ANIN} and make $V_{IN2} = V_{OFS}$.

We would then have, from IV:

$$V_{IN} = V_{ANIN}, V_{IN2} = V_{OFS}$$

However, using a 256-resistor chain to provide references, we can find a V_{IN} , V_{IN2} combination which can bring V_{OUT} close to the value of V_{BIAS} . The 256-resistor chain provides a reference voltage in 20 mV steps. We can then take separate taps of the resistor chain and connect them to V_{IN} and V_{IN2} . The voltage attached to V_{IN} will couple to V_{OUT} by a factor of $C_A/(C_A + C_B) = 8/9$ from EQN IV. The voltage attached to V_{IN2} will couple to V_{OUT} by a factor of $C_B/(C_A + C_B)$. The ratio of the impacts on V_{OUT} of V_{IN} versus V_{IN2} is:

$$\left(\frac{\partial V_{OUT}}{\partial V_{IN}} \right) \div \left(\frac{\partial V_{OUT}}{\partial V_{IN2}} \right) = (8/9)/(1/9) = 8$$

Therefore, a voltage change on V_{IN} will affect the voltage seen at V_{OUT} eight times more than the same change placed on V_{IN2} .

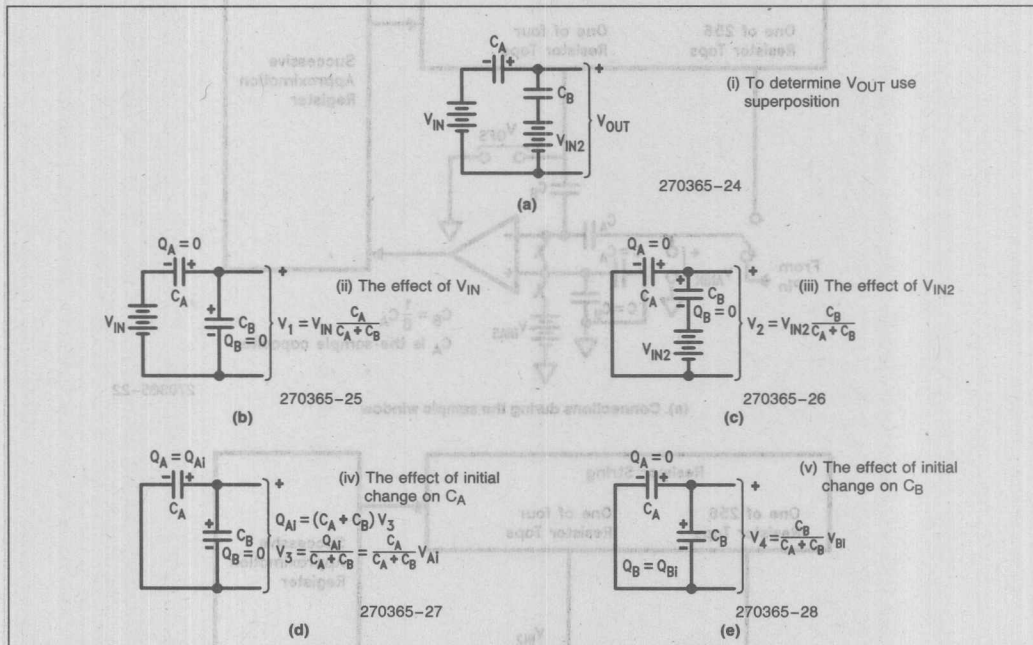


Figure B2. Superposition Analysis of comparator input voltage

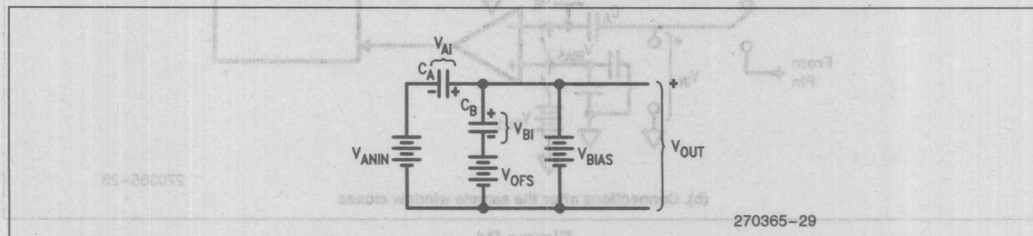


Figure B3. Initial Conditions

For example, assume the actual input voltage V_{ANIN} was 2.50mV during the sample window. Using EQN IV, and assuming $V_{BIAS} = 3V$ and $V_{OFS} = 70mV$, we substitute and find:

$$V_{OUT} = (V_{IN} + 2.9975) \times (8/9) + (V_{IN2} + 2.93) \times (1/9) \quad (V)$$

Using successive approximation, the first trial input voltage attempted corresponds to the digital code 0111 1111 11b ($127 \times 20mV + 10mV$). This means that the voltage applied to V_{IN} will be the 0111 1111b tap and the voltage applied to V_{IN2} will be the 0110b tap ($6 \times 20mV + 10mV = 3 \text{ LSB}$). Substituting these values into EQN V we have:

$$V_{OUT} = (2.550 + 2.9975) \times (8/9) + (0.130 + 2.93) \times (1/9) \quad (V)$$

$$V_{OUT} = 4.931 + 0.34 = 5.271$$

Since the 3V reference is lower than V_{OUT} with these inputs, the comparator will output a 0 which is placed in the MSB of the successive approximation register. The next most significant bit of the SAR is then zero'd

and the new ladder tap applied to V_{IN} . The result of this second comparison, and the subsequent comparisons are shown in Table B1. The C program used to generate Table B1 is listed in Listing B1.

The value selected for V_{OFS} during the sample window may not be obvious. The purpose of V_{OFS} is to inject a constant offset in the sampling process so that the converter's first code transition will occur at 2.5mV.

Using EQN IV we can quickly see why V_{OFS} is chosen to be the fourth resistor tap ($4 \times 20mV + 10mV = 70mV$). For $V_{ANIN} = 2.5mV$, we want V_{OUT} to evaluate to V_{BIAS} when the SAR is OH.

$$V_{OUT} = \{(0.20 \text{ mV} + 10 \text{ mV}) + (V_{BIAS} - 2.5 \text{ mV})\} \times (8/9) + \{(0.20 \text{ mV} + 10 \text{ mV}) + (V_{BIAS} - 70 \text{ mV})\} \times (1/9)$$

$$V_{OUT} - V_{BIAS} = 7.5 \text{ mV} \times (8/9) - 60 \text{ mV} \times (1/9) = 0$$

Therefore, if $V_{OFS} = 70 \text{ mV}$, the converter's first code transition will be when $V_{ANIN} = 2.5 \text{ mV}$.

Table B1. Conversion Simulation

A to D simulator. (center taps) . . With		
$V_{IN} = 0.002500$		
$V_{CENT} = 3.000000 \quad V_{OFF} = 0.070000$		
SAR = 1FFH (511)	$V_{OUT} = 5.271111$	
SAR = FFH (255)	$V_{OUT} = 4.133333$	
SAR = 7FH (127)	$V_{OUT} = 3.564444$	
SAR = 3FH (63)	$V_{OUT} = 3.280000$	
SAR = 1FH (31)	$V_{OUT} = 3.137778$	
SAR = FH (15)	$V_{OUT} = 3.066667$	
SAR = 7H (7)	$V_{OUT} = 3.031111$	
SAR = 3H (3)	$V_{OUT} = 3.013333$	
SAR = 1H (1)	$V_{OUT} = 3.004444$	
SAR = 0H (0)	$V_{OUT} = 3.000000$	
SAR = 1H (1)	which means 0.005000 volts	

```

#include "STDIO.H"
/* example invocation lines

a2dsin 0.0025 3.0 0.07 p
Vin Vbias Vofs print to screen and lp

a2dsin 0.0075 3.0 0.07
Vin Vbias Vofs print to screen only

*/

int main(k, argv)
int k;
char *argv[];
{
    FILE *fp, *fopen();
    double initial_conditions, vin, vout, vcent, voff, v89, v19;
    unsigned int sar = 0x3FF;
    unsigned int mask = 0x200;
    unsigned int count = 0;
    unsigned int printon;
    if (strcmp(argv[0], "run") == 0)
        count++;
    if ((k != (4 + count)) & (k != (5 + count)))
    {
        printf("\nInvocation error!\n");
        return;
    }
    count++;
    sscanf(argv[count++], "%lf", &vin);
    sscanf(argv[count++], "%lf", &vcent);
    sscanf(argv[count++], "%lf", &voff);
    if (count == k)
        printon = 0;
    else printon = 1;
    printf("A to D simulator.(center taps)..");
    if (printon)
    {
        if ((fp = fopen("lprn:", "w")) == 0)
        {
            printf("\nCan't open printer\n");
            return;
        }
    }
    if (printon)
        fprintf(fp, "A to D simulator..");
    printf(" with \nVin = %f\nVcent = %f\nVoff = %f\n", vin, vcent, voff);
    if (printon)
        fprintf(fp, " with \nVin = %f\nVcent = %f\nVoff = %f\n",
            vin, vcent, voff);

    initial_conditions = ((8.0 / 9.0) * (vcent - vin))
        + ((1.0 / 9.0) * (vcent - voff));
    v89 = 8.0 / 9.0;
    v19 = 1.0 / 9.0;
}

```

Listing B1. A/D Converter Simulator

```

sar ^= mask;
printf("SAR = %3xH (%4d)\t", sar, sar);
if (printon)
    fprintf(fp, "SAR = %3xH (%4d)\t", sar, sar);
for (count = 0; count < 10; count++)
{
    vout = (v89 * (((double) (sar >> 2)) * 0.02 + 0.01))
        + (v19 * (((double) ((sar & 3) << 1)) * 0.02 + 0.01))
        + initial_conditions;
    if (vout < vcent)
        sar |= mask;
    mask >>= 1;
    sar ^= mask;
    printf("Vout = %f\nSAR = %3xH (%4d)\t", vout, sar, sar);
    if (printon)
        fprintf(fp, "Vout = %f\nSAR = %3xH (%4d)\t",
            vout, sar, sar);
}
printf(" which means %f volts\n", (double) sar * 0.005);
if (printon)
    fprintf(fp, " which means %f volts\n", (double) sar * 0.005);
return;
}
/* main */

```

270365-A6

Listing B1: A/D Converter Simulator (Continued)

APPENDIX C ERROR FORMULAS

The following C program listing contains the routines used to calculate A/D performance in the Embedded Controller Applications lab. Most of the routines require floating point arrays to operate upon. In the listings, the array `x[]` contains the input voltages corresponding to each code transition of the converter. The array `dx[]` contains the width of the region in which each code transition of the converter could occur. For example, an input voltage of 0.003V may cause code 0 and code 1 to be equally likely outputs. `x[0]` would then contain 0.0030000. However, 0-to-1 code transitions might be observed infrequently through a range of input voltages from 0.0025V to 0.0035V. `dx[0]` would then contain 0.0010000 to indicate that there is a 1 millivolt window in which either code could occur. `x[]` and `dx[]` are generated by hardware doing repeat-

ed conversions using precision voltage standards to provide the input voltages. The array `dd[]` is used throughout as temporary storage.

Generally, typical data is drawn from `x[]` only. When minimum and maximum data is desired, `x[]` and `dx[]` are used to find the range of possible input voltages that could cause each code. For example, typical zero offset is found by simply subtracting 0.5 LSB from the value of `x[0]`. But, the minimum and maximum zero offset would be calculated as $x[0] - 0.5 \text{ LSB} \pm dx[0]/2$.

The listings are provided to show exactly how performance data is calculated. They are not meant to be compiled by the reader. In fact, they are too incomplete to compile correctly, as some support routines and global data structures are not provided.


```

#include "DPR\ADTMAC.H"
#include "DPR\YDBASE.H"
#include "DPR\RDBASE.H"
#define LSB (now.avcc/(pow(2,nbits)))
#define FCT (int)(pow(2,nbits) - 2)
#undef min
#undef max
#undef abs

double pow(a, b)
int a, b;
{
    double temp;
    int i;
    temp = 1.0;
    for (i = 1; i <= ((int) b); i++, temp = temp * a)
        ;
    return (temp);
}

double fabs(a)
double a;
{
    if (a < 0)
        return (-a);
    else return (a);
}

int min(a, b)
double a, b;
{
    if (a < b)
        return (1);
    else if (a > b)
        return (2);
    else return (0);
}

int max(a, b)
double a, b;
{
    return (min(b, a));
}

double typeoff(x, dx)
float x[], dx[];
{
    double pov();
    return (x[0] - (0.5 * LSB));
}

double maxoff(x, dx)
float x[], dx[];
{
    double pov();
    return (x[0] + (dx[0] / 2.0) - 0.5 * LSB);
}

double minoff(x, dx)

```

270365-A7

Listing C1. Error Formulas

```

float x[], dx[];
{
    double pow();
    return (x[0] - (dx[0] / 2.0) - 0.5 * LSB);
}

double typfse(x, dx)
float x[], dx[];
{
    double pow();
    return (x[FCT] - (now.avcc - (1.5 * LSB)));
}

double minfse(x, dx)
float x[], dx[];
{
    double pow();
    return ((x[FCT] - (dx[FCT] / 2.0)) - (now.avcc - (1.5 * LSB)));
}

double maxfse(x, dx)
float x[], dx[];
{
    double pow();
    return ((x[FCT] + (dx[FCT] / 2.0)) - (now.avcc - (1.5 * LSB)));
}

int xaberror(x, dx, dd, start, stop) /* transition absolute error */
float x[], dx[], dd[];
unsigned int start, stop;
{
    double pow(), fabs();
    int i, worst;
    for (i = worst = start; i <= stop; i++)
    {
        dd[i] = x[i] - (((double) i + 0.5) * LSB);
        if (fabs(dd[i]) > fabs(dd[worst]))
            worst = i;
    }
    return (worst);
}

int xaberrordx(x, dx, dd, start, stop) /* transition absolute error w/dx */
float x[], dx[], dd[];
unsigned int start, stop;
{
    double pow(), fabs();
    int i, worst;
    double t1, t2;
    for (i = worst = start; i <= stop; i++)
    {
        t1 = (x[i] - (dx[i] / 2.0)) - (((double) i + 0.5) * LSB);
        t2 = (x[i] + (dx[i] / 2.0)) - (((double) i + 0.5) * LSB);
        if (fabs(t1) > fabs(t2))
            dd[i] = t1;
        else dd[i] = t2;
        if (fabs(dd[i]) > fabs(dd[worst]))
            worst = i;
    }
    return (worst);
}

```

Listing C1. Error Formulas (Continued)

270365-A8

```

int tbnonlin(x, dx, dd, start, stop) /* tb nonlin using x only */
float x[], dx[], dd[]:
unsigned int start, stop;
{
    int i, worst;
    double pow(), typsoff(), typfse(), fabs();
    double oadj, qadj;

    oadj = typsoff(x, dx);
    qadj = 1.0 + ((typfse(x, dx) - oadj) / x[stop]);

    for (i = worst = start; i <= stop; i++)
    {
        dd[i] = (x[i] - oadj) * qadj - (((double) 1 + 0.5) * LSB);
        if (fabs(dd[i]) > fabs(dd[worst]))
            worst = i;
    }
    return (worst);
}

int tbnonlindx(x, dx, dd, start, stop) /* tb nonlin using x and dx */
float x[], dx[], dd[]:
unsigned int start, stop;
{
    int i, worst;
    double pow(), typsoff(), typfse(), fabs();
    double oadj, qadj, t1, t2;

    oadj = typsoff(x, dx);
    qadj = 1.0 + ((typfse(x, dx) - oadj) / x[stop]);

```

270365-A9

Listing C1. Error Formulas (Continued)

```

    for (i = worst = start; i != stop; i++)
    {
        t1 = (x[i] - (dx[i] / 2.0) - oadj) * gadj - (((double) i + 0.5) * LSB);
        t2 = (x[i] + (dx[i] / 2.0) - oadj) * gadj - (((double) i + 0.5) * LSB);
        if (fabs(t1) > fabs(t2))
            dd[i] = t1;
        else dd[i] = t2;
        if (fabs(dd[i]) > fabs(dd[worst]))
            worst = i;
    }
    return (worst);
}

int xdn1(x, dx, dd, start, stop) /* using x only */
float x[], dx[], dd[];
int start, stop;
{
    int i, worst;
    double pov(), fabs();
    double oadj, gadj;
    double typfse(), typzoff();

    oadj = typzoff(x, dx);
    gadj = 1.0 + ((typfse(x, dx) - oadj) / x[stop]);

    worst = start;
    if (start == 0)
    {
        dd[0] = 0.0;
        start++;
    }
    for (i = start; i != stop; i++)
    {
        dd[i] = (x[i] - oadj) * gadj
            - (x[i - 1] - oadj) * gadj
            - LSB;
        if (fabs(dd[i]) > fabs(dd[worst]))
            worst = i;
    }
    return (worst);
}

int xdn1dx(x, dx, dd, start, stop) /* using x and dx */
float x[], dx[], dd[];
int start, stop;
{
    int i, worst;
    double pov(), fabs();
    double t1, t2;
    double oadj, gadj;
    double typfse(), typzoff();

    oadj = typzoff(x, dx);
    gadj = 1.0 + ((typfse(x, dx) - oadj) / x[stop]);

    worst = start;
    if (start == 0)
    {
        dd[0] = dx[0] / 2.0;

```

270365-B0

Listing C1. Error Formulas (Continued)

APPENDIX D

SAMPLE CONVERTER DATA

The following pages include printouts describing the performance of an 8097BH. The data shown is for one device and is provided for illustrative purposes only. Users should only rely upon data sheet specifications for the exact device they are designing with.

Table D1 summarizes many performance measures for one converter at 25 C, 12 MHz, $V_{CC} = 5.00$ volts and

$V_{REF} = 5.120$ volts. Following Table D2 are several error plots that describe Absolute Error, Terminal-based Non-Linearity, Differential Non-Linearity and Repeatability for the test device code-by-code. The y-axis in the plots is the error in volts for each code transition, where code transitions make up the x-axis.

Table D1. Sample Converter Data

```

Test ID = DOH
sN: 4130 (1022H)
T = 25.000000
VCC = 5.000000, Avcc = 5.120000
Freq = 12.000000
Chan. = 3
States = 188 Mode = OH
X0.15 1/28/87
Transition Characterization Parameter Listing
Large Step = 0.001000 V
Small Step = 0.000100 V
Endpoints when (1/100) are wrong

Center is 50 percent

Typical Offset Error = -0.001923
Maximum Offset Error = -0.002460
Maximum Offset Error = -0.001385

Typical FS Error = -0.000566
Maximum FS Error = -0.001254
Minimum FS Error = -0.000120

Absolute Error (typ) 40 = 0.004157
Absolute Error (max) 40 = 0.004795
Absolute Error (min) 325 = 0.001111

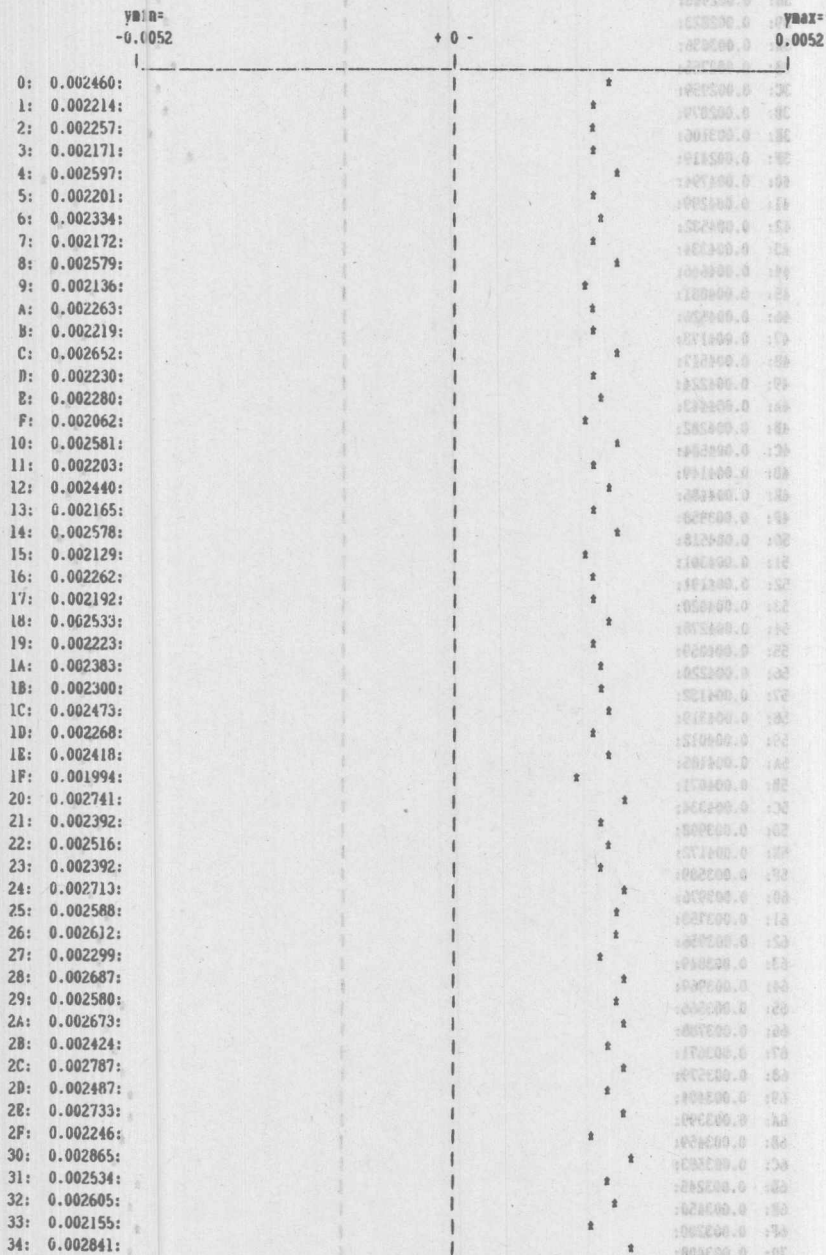
Diff. Non. Lin. Error (max) 40 = 0.003747
Diff. Non. Lin. Error (min) FF = -0.001071

Term. Non. Lin. Error (max) 325 = -0.004102
Term. Non. Lin. Error (min) 40 = 0.002148

Maximum Reliability Error 3D1 = 0.001875
Minimum Reliability Error 3A7 = 0.000974

Resolution is 1024 levels.
  
```

Absolute Error, SN = 4130



01-28075

270365-69

(beun) Absolute Error, SN = 4130 (edA)

[illegible]

Absolute Error, SN = 4130 (Continued)

71:	0.003203:	5P2100.0	2A
72:	0.003238:	1ST100.0	2A
73:	0.003201:	8C2100.0	2A
74:	0.003281:	30P100.0	2B
75:	0.002882:	3CT100.0	1B
76:	0.003161:	72B100.0	2B
77:	0.003112:	HTT100.0	2A
78:	0.003000:	282100.0	2A
79:	0.002833:	892100.0	2B
7A:	0.002989:	100100.0	2B
7B:	0.002932:	922100.0	2B
7C:	0.002924:	21T100.0	2B
7D:	0.002716:	142100.0	2B
7E:	0.002759:	823100.0	2A
7F:	0.002027:	1AT100.0	2B
80:	0.003422:	109100.0	2A
81:	0.003129:	982100.0	2B
82:	0.003322:	882100.0	2A
83:	0.003169:	038100.0	2B
84:	0.003202:	163100.0	2B
85:	0.002953:	11P200.0	2B
86:	0.003086:	04P200.0	2B
87:	0.002897:	9TT100.0	2B
88:	0.003038:	108200.0	2B
89:	0.002446:	71T200.0	2B
8A:	0.002983:	16P200.0	2B
8B:	0.002623:	108500.0	2B
8C:	0.002813:	100100.0	2B
8D:	0.002593:	8PT200.0	2B
8E:	0.002485:	128500.0	2A
8F:	0.002415:	288500.0	2B
90:	0.002791:	086600.0	2B
91:	0.002647:	87T200.0	2B
92:	0.002812:	878500.0	2B
93:	0.002576:	15AT200.0	2B
94:	0.002682:	128800.0	2B
95:	0.002514:	048500.0	2B
96:	0.002711:	06T200.0	2B
97:	0.002405:	106500.0	2B
98:	0.002593:	842500.0	2B
99:	0.002268:	100100.0	2B
9A:	0.002550:	83T200.0	2B
9B:	0.002340:	80T200.0	2B
9C:	0.002412:	188500.0	2B
9D:	0.002118:	71100.0	2B
9E:	0.002303:	120T200.0	2B
9F:	0.001754:	218500.0	2B
A0:	0.002191:	088200.0	2B
A1:	0.001893:	102100.0	2B
A2:	0.002259:	12T200.0	2B
A3:	0.001986:	02T200.0	2B
A4:	0.002103:	128100.0	2B
A5:	0.001881:	10100.0	2B
A6:	0.002071:	8C100.0	2B
A7:	0.001933:	101500.0	2B
A8:	0.002059:	128200.0	2B
A9:	0.001792:	121000.0	2B
AA:	0.001967:	101500.0	2B
AB:	0.001776:	101500.0	2B
AC:	0.001864:	101500.0	2B

ST-388011

270365-71

AD: 0.001592:		*:00000.0 :1T
AE: 0.001781:		*:00000.0 :1T
AF: 0.001538:		*:00000.0 :1T
AG: 0.001906:		*:00000.0 :1T
AH: 0.001724:		*:00000.0 :1T
AI: 0.001887:		*:00000.0 :1T
AJ: 0.001773:		*:00000.0 :1T
AK: 0.001585:		*:00000.0 :1T
AL: 0.001598:		*:00000.0 :1T
AM: 0.001650:		*:00000.0 :1T
AN: 0.001554:		*:00000.0 :1T
AO: 0.001715:		*:00000.0 :1T
AP: 0.001545:		*:00000.0 :1T
AQ: 0.001653:		*:00000.0 :1T
AR: 0.001474:		*:00000.0 :1T
AS: 0.001467:		*:00000.0 :1T
AT: 0.001384:		*:00000.0 :1T
AV: 0.001588:		*:00000.0 :1T
AW: 0.001020:		*:00000.0 :1T
AX: 0.003214:		*:00000.0 :1T
AY: 0.002914:		*:00000.0 :1T
AZ: 0.002966:		*:00000.0 :1T
BA: 0.002779:		*:00000.0 :1T
BB: 0.003087:		*:00000.0 :1T
BC: 0.002717:		*:00000.0 :1T
BD: 0.003096:		*:00000.0 :1T
BE: 0.002806:		*:00000.0 :1T
BF: 0.003030:		*:00000.0 :1T
BG: 0.002796:		*:00000.0 :1T
BH: 0.002642:		*:00000.0 :1T
BI: 0.002885:		*:00000.0 :1T
BJ: 0.003040:		*:00000.0 :1T
BK: 0.002719:		*:00000.0 :1T
BL: 0.002878:		*:00000.0 :1T
BM: 0.002742:		*:00000.0 :1T
BN: 0.002845:		*:00000.0 :1T
BO: 0.002546:		*:00000.0 :1T
BP: 0.002790:		*:00000.0 :1T
BQ: 0.002395:		*:00000.0 :1T
BR: 0.002848:		*:00000.0 :1T
BS: 0.002487:		*:00000.0 :1T
BT: 0.002768:		*:00000.0 :1T
BU: 0.002700:		*:00000.0 :1T
BV: 0.002681:		*:00000.0 :1T
BW: 0.002617:		*:00000.0 :1T
BX: 0.002755:		*:00000.0 :1T
BY: 0.002643:		*:00000.0 :1T
BZ: 0.002684:		*:00000.0 :1T
CA: 0.002398:		*:00000.0 :1T
CB: 0.002553:		*:00000.0 :1T
CC: 0.002223:		*:00000.0 :1T
CD: 0.002403:		*:00000.0 :1T
CE: 0.001878:		*:00000.0 :1T
CF: 0.002439:		*:00000.0 :1T
CG: 0.002206:		*:00000.0 :1T
CH: 0.002083:		*:00000.0 :1T
CI: 0.002055:		*:00000.0 :1T
CJ: 0.002288:		*:00000.0 :1T
CK: 0.002144:		*:00000.0 :1T
CL: 0.002356:		*:00000.0 :1T

IT-288073

270365-72

Absolute Error, SN = 4130 (Continued)

E9:	0.002225:	1057000.0	*151
EA:	0.002263:	1029000.0	*151
EB:	0.002113:	1079000.0	*151
EC:	0.002233:	1000100.0	*151
ED:	0.002172:	1029000.0	*151
EE:	0.002369:	107100.0	*151
EF:	0.002149:	1019100.0	*151
F0:	0.002216:	1009000.0	*151
F1:	0.001841:	1000300.0	*151
F2:	0.002051:	1010000.0	*151
F3:	0.001935:	1010000.0	*151
F4:	0.001965:	1029000.0	*151
F5:	0.001729:	1070000.0	*151
F6:	0.001979:	1010000.0	*151
F7:	0.001899:	1020000.0	*151
F8:	0.001589:	1050000.0	*151
F9:	0.001718:	1010000.0	*151
FA:	0.001935:	1000000.0	*151
FB:	0.001756:	1020000.0	*151
FC:	0.001975:	1000100.0	*151
FD:	0.001832:	1020000.0	*151
FE:	0.001920:	1000000.0	*151
FF:	0.001041:	1000000.0	*151
100:	0.002291:	1000100.0	*151
101:	0.002008:	1000000.0	*151
102:	0.002296:	1000000.0	*151
103:	0.001975:	1000000.0	*151
104:	0.001946:	1010000.0	*151
105:	0.001874:	1000000.0	*151
106:	0.001884:	1000000.0	*151
107:	0.001817:	1000000.0	*151
108:	0.002135:	1000000.0	*151
109:	0.001921:	1000000.0	*151
10A:	0.002009:	1000000.0	*151
10B:	0.001832:	1000000.0	*151
10C:	0.001903:	1000000.0	*151
10D:	0.001694:	1000000.0	*151
10E:	0.001838:	1000000.0	*151
10F:	0.001537:	1000000.0	*151
110:	0.001681:	1000000.0	*151
111:	0.001436:	1000000.0	*151
112:	0.001730:	1000000.0	*151
113:	0.001631:	1000000.0	*151
114:	0.001636:	1000000.0	*151
115:	0.001374:	1000000.0	*151
116:	0.001550:	1000000.0	*151
117:	0.001500:	1000000.0	*151
118:	0.001530:	1000000.0	*151
119:	0.001411:	1000000.0	*151
11A:	0.001390:	1000000.0	*151
11B:	0.001271:	1000000.0	*151
11C:	0.001321:	1000000.0	*151
11D:	0.001074:	1000000.0	*151
11E:	0.001268:	1000000.0	*151
11F:	0.000814:	1000000.0	*151
120:	0.001401:	1000000.0	*151
121:	0.001052:	1000000.0	*151
122:	0.001193:	1000000.0	*151
123:	0.001106:	1000000.0	*151
124:	0.001253:	1000000.0	*151

AT-2000-73

270365-73

Absolute Error, SN = 4130 (Continued)

125:	0.000758:		*55500.0	109
126:	0.000953:		*55500.0	108
127:	0.000976:		*55500.0	106
128:	0.001080:		*55500.0	105
129:	0.000937:		*55500.0	102
12A:	0.001181:		*55500.0	100
12B:	0.001018:		*55500.0	99
12C:	0.000959:		*55500.0	97
12D:	0.000862:		*55500.0	95
12E:	0.000812:		*55500.0	93
12F:	0.000813:		*55500.0	92
130:	0.000933:		*55500.0	91
131:	0.000671:		*55500.0	89
132:	0.000811:		*55500.0	88
133:	0.000634:		*55500.0	87
134:	0.000929:		*55500.0	86
135:	-0.000647:		*55500.0	85
136:	0.000888:		*55500.0	84
137:	0.000539:		*55500.0	83
138:	0.001027:		*55500.0	82
139:	0.000850:		*55500.0	81
13A:	0.000749:		*55500.0	80
13B:	0.000809:		*55500.0	79
13C:	0.001032:		*55500.0	78
13D:	0.000788:		*55500.0	77
13E:	0.000963:		*55500.0	76
13F:	-0.000681:		*55500.0	75
140:	0.002218:		*55500.0	74
141:	0.002186:		*55500.0	73
142:	0.002327:		*55500.0	72
143:	0.002196:		*55500.0	71
144:	0.002447:		*55500.0	70
145:	0.002267:		*55500.0	69
146:	0.002435:		*55500.0	68
147:	0.002385:		*55500.0	67
148:	0.002554:		*55500.0	66
149:	0.002284:		*55500.0	65
14A:	0.002420:		*55500.0	64
14B:	0.002482:		*55500.0	63
14C:	0.002523:		*55500.0	62
14D:	0.002299:		*55500.0	61
14E:	0.002303:		*55500.0	60
14F:	0.002097:		*55500.0	59
150:	0.002267:		*55500.0	58
151:	0.002127:		*55500.0	57
152:	0.002312:		*55500.0	56
153:	0.002092:		*55500.0	55
154:	0.002264:		*55500.0	54
155:	0.001976:		*55500.0	53
156:	0.002034:		*55500.0	52
157:	0.002084:		*55500.0	51
158:	0.002235:		*55500.0	50
159:	0.001959:		*55500.0	49
15A:	0.002071:		*55500.0	48
15B:	0.002048:		*55500.0	47
15C:	0.002104:		*55500.0	46
15D:	0.001998:		*55500.0	45
15E:	0.002110:		*55500.0	44
15F:	0.001935:		*55500.0	43
160:	0.002075:		*55500.0	42

270365-74

270365-74

Absolute Error, SN = 4130 (Continued)

[illegible]

6

19D: 0.001262:	1827100.0 1181
19E: 0.001245:	1828100.0 1581
19F: 0.001201:	1829100.0 1581
1A0: 0.001413:	1830100.0 1881
1A1: 0.001170:	1831100.0 1281
1A2: 0.001361:	1832100.0 1081
1A3: 0.001321:	1833100.0 1781
1A4: 0.001181:	1834100.0 1881
1A5: 0.000872:	1835100.0 1981
1A6: 0.001086:	1836100.0 1881
1A7: 0.001080:	1837100.0 1581
1A8: 0.001195:	1838100.0 1981
1A9: 0.001138:	1839100.0 1881
1AA: 0.001204:	1840100.0 1981
1AB: 0.001230:	1841100.0 1881
1AC: 0.001210:	1842100.0 1881
1AD: 0.000971:	1843100.0 1581
1AE: 0.001083:	1844100.0 1581
1AF: 0.001274:	1845100.0 1581
1B0: 0.001211:	1846100.0 1581
1B1: 0.001133:	1847100.0 1581
1B2: 0.001069:	1848100.0 1881
1B3: 0.001095:	1849100.0 1781
1B4: 0.001065:	1850100.0 1581
1B5: 0.001081:	1851100.0 1881
1B6: 0.001124:	1852100.0 1881
1B7: 0.001079:	1853100.0 1881
1B8: 0.001040:	1854100.0 1881
1B9: 0.001081:	1855100.0 1881
1BA: 0.001183:	1856100.0 1881
1BB: 0.001297:	1857100.0 1881
1BC: 0.001124:	1858100.0 1881
1BD: 0.001006:	1859100.0 1881
1BE: 0.001046:	1860100.0 1881
1BF: 0.001061:	1861100.0 1881
1C0: 0.002475:	1862100.0 1881
1C1: 0.002358:	1863100.0 1881
1C2: 0.002538:	1864100.0 1881
1C3: 0.002457:	1865100.0 1881
1C4: 0.002712:	1866100.0 1881
1C5: 0.002415:	1867100.0 1881
1C6: 0.002579:	1868100.0 1881
1C7: 0.002436:	1869100.0 1881
1C8: 0.002796:	1870100.0 1881
1C9: 0.002388:	1871100.0 1881
1CA: 0.002368:	1872100.0 1881
1CB: 0.002426:	1873100.0 1881
1CC: 0.002661:	1874100.0 1881
1CD: 0.002462:	1875100.0 1881
1CE: 0.002497:	1876100.0 1881
1CF: 0.002396:	1877100.0 1881
1D0: 0.002617:	1878100.0 1881
1D1: 0.002399:	1879100.0 1881
1D2: 0.002503:	1880100.0 1881
1D3: 0.002453:	1881100.0 1881
1D4: 0.002623:	1882100.0 1881
1D5: 0.002414:	1883100.0 1881
1D6: 0.002423:	1884100.0 1881
1D7: 0.002490:	1885100.0 1881
1D8: 0.002606:	1886100.0 1881

ST-286013

270365-76

Absolute Error, SN = 4130 (Continued)

1D9: 0.002351:	1501100.0 *115
1DA: 0.002439:	1521100.0 *115
1DB: 0.002382:	1551100.0 *115
1DC: 0.002426:	1581100.0 *115
1DD: 0.002376:	1611100.0 *115
1DE: 0.002443:	1641100.0 *115
1DF: 0.002531:	1671100.0 *115
1EO: 0.002583:	1701100.0 *115
1EI: 0.002038:	1731100.0 *115
1E2: 0.002371:	1761100.0 *115
1E3: 0.002043:	1791100.0 *115
1E4: 0.002350:	1821100.0 *115
1E5: 0.002166:	1851100.0 *115
1E6: 0.002351:	1881100.0 *115
1E7: 0.002363:	1911100.0 *115
1E8: 0.002455:	1941100.0 *115
1E9: 0.002002:	1971100.0 *115
1EA: 0.002299:	2001100.0 *115
1EB: 0.002146:	2031100.0 *115
1EC: 0.002279:	2061100.0 *115
1ED: 0.002072:	2091100.0 *115
1EE: 0.001960:	2121100.0 *115
1EF: 0.002221:	2151100.0 *115
1FO: 0.002314:	2181100.0 *115
1F1: 0.001940:	2211100.0 *115
1F2: 0.002086:	2241100.0 *115
1F3: 0.002310:	2271100.0 *115
1F4: 0.002188:	2301100.0 *115
1F5: 0.002075:	2331100.0 *115
1F6: 0.002065:	2361100.0 *115
1F7: 0.002267:	2391100.0 *115
1F8: 0.002187:	2421100.0 *115
1F9: 0.002002:	2451100.0 *115
1FA: 0.002120:	2481100.0 *115
1FB: 0.002133:	2511100.0 *115
1FC: 0.002158:	2541100.0 *115
1FD: 0.001937:	2571100.0 *115
1FE: 0.002079:	2601100.0 *115
1FF: 0.001409:	2631100.0 *115
200: 0.001879:	2661100.0 *115
201: 0.001707:	2691100.0 *115
202: 0.001905:	2721100.0 *115
203: 0.001557:	2751100.0 *115
204: 0.001650:	2781100.0 *115
205: 0.001661:	2811100.0 *115
206: 0.001683:	2841100.0 *115
207: 0.001595:	2871100.0 *115
208: 0.001535:	2901100.0 *115
209: 0.001179:	2931100.0 *115
20A: 0.001610:	2961100.0 *115
20B: 0.001454:	2991100.0 *115
20C: 0.001370:	3021100.0 *115
20D: 0.001262:	3051100.0 *115
20E: 0.001179:	3081100.0 *115
20F: 0.000983:	3111100.0 *115
210: 0.001405:	3141100.0 *115
211: 0.001074:	3171100.0 *115
212: 0.001168:	3201100.0 *115
213: 0.001193:	3231100.0 *115
214: 0.001420:	3261100.0 *115

215:	0.001162:	124500.0	:901
216:	0.001323:	100100.0	:891
217:	0.001268:	100100.0	:891
218:	0.001296:	100100.0	:901
219:	0.001147:	100100.0	:891
21A:	0.001036:	100100.0	:891
21B:	0.001170:	100100.0	:901
21C:	0.001551:	100100.0	:901
21D:	0.001065:	100100.0	:101
21E:	0.001216:	100100.0	:901
21F:	0.000666:	100100.0	:901
220:	0.001304:	100100.0	:901
221:	0.000988:	100100.0	:901
222:	0.001207:	100100.0	:901
223:	0.001066:	100100.0	:901
224:	0.001079:	100100.0	:901
225:	0.001029:	100100.0	:901
226:	0.000971:	100100.0	:901
227:	0.000968:	100100.0	:901
228:	0.001203:	100100.0	:901
229:	0.000949:	100100.0	:901
22A:	0.001026:	100100.0	:901
22B:	0.001051:	100100.0	:901
22C:	0.001118:	100100.0	:901
22D:	0.000887:	100100.0	:901
22E:	0.001149:	100100.0	:901
22F:	0.000738:	100100.0	:901
230:	0.001214:	100100.0	:901
231:	0.000920:	100100.0	:901
232:	0.001203:	100100.0	:901
233:	0.000978:	100100.0	:901
234:	0.001203:	100100.0	:901
235:	0.001081:	100100.0	:901
236:	0.001003:	100100.0	:901
237:	0.001053:	100100.0	:901
238:	0.001235:	100100.0	:901
239:	0.000705:	100100.0	:901
23A:	0.001066:	100100.0	:901
23B:	0.000924:	100100.0	:901
23C:	0.001087:	100100.0	:901
23D:	0.001000:	100100.0	:901
23E:	0.001006:	100100.0	:901
23F:	-0.000785:	100100.0	:901
240:	0.002137:	100100.0	:901
241:	0.001968:	100100.0	:901
242:	0.002196:	100100.0	:901
243:	0.002027:	100100.0	:901
244:	0.002162:	100100.0	:901
245:	0.001918:	100100.0	:901
246:	0.002075:	100100.0	:901
247:	0.001871:	100100.0	:901
248:	0.002060:	100100.0	:901
249:	0.002108:	100100.0	:901
24A:	0.002100:	100100.0	:901
24B:	0.002060:	100100.0	:901
24C:	0.002217:	100100.0	:901
24D:	0.002035:	100100.0	:901
24E:	0.002245:	100100.0	:901
24F:	0.002190:	100100.0	:901
250:	0.002415:	100100.0	:901

11-280015

270365-78

Absolute Error, SN = 4130 (Continued)

251: 0.002013:	:03400.0 * :05
252: 0.002259:	:002100.0 * :05
253: 0.002068:	:002100.0 * :05
254: 0.002370:	:002100.0 * :05
255: 0.002213:	:034100.0 * :05
256: 0.002314:	:034100.0 * :05
257: 0.002207:	:002100.0 * :05
258: 0.002259:	:034100.0 * :05
259: 0.002090:	:121100.0 * :05
25A: 0.001956:	:002100.0 * :05
25B: 0.002095:	:112100.0 * :05
25C: 0.002377:	:002100.0 * :05
25D: 0.002086:	:002100.0 * :05
25E: 0.002090:	:002100.0 * :05
25F: 0.001972:	:002100.0 * :05
260: 0.002137:	:002100.0 * :05
261: 0.001808:	:002000.0 * :05
262: 0.002022:	:002100.0 * :05
263: 0.001944:	:002000.0 * :05
264: 0.002053:	:002000.0 * :05
265: 0.001856:	:002000.0 * :05
266: 0.002042:	:002100.0 * :05
267: 0.001940:	:002000.0 * :05
268: 0.002020:	:002000.0 * :05
269: 0.001762:	:002000.0 * :05
26A: 0.001820:	:002000.0 * :05
26B: 0.001773:	:002000.0 * :05
26C: 0.001850:	:002100.0 * :05
26D: 0.001685:	:002000.0 * :05
26E: 0.001910:	:002000.0 * :05
26F: 0.001794:	:002000.0 * :05
270: 0.001748:	:002000.0 * :05
271: 0.001653:	:002000.0 * :05
272: 0.001632:	:002000.0 * :05
273: 0.001540:	:002000.0 * :05
274: 0.001677:	:002000.0 * :05
275: 0.001356:	:002000.0 * :05
276: 0.001582:	:002000.0 * :05
277: 0.001630:	:002000.0 * :05
278: 0.001505:	:002000.0 * :05
279: 0.001403:	:002000.0 * :05
27A: 0.001464:	:002000.0 * :05
27B: 0.001402:	:002000.0 * :05
27C: 0.001620:	:002000.0 * :05
27D: 0.001106:	:002000.0 * :05
27E: 0.001437:	:002000.0 * :05
27F: 0.001276:	:002000.0 * :05
280: 0.001913:	:002000.0 * :05
281: 0.001950:	:002000.0 * :05
282: 0.002095:	:002000.0 * :05
283: 0.001620:	:002000.0 * :05
284: 0.002096:	:002000.0 * :05
285: 0.001850:	:002100.0 * :05
286: 0.001951:	:002100.0 * :05
287: 0.001836:	:002000.0 * :05
288: 0.001726:	:002100.0 * :05
289: 0.001690:	:002100.0 * :05
28A: 0.001743:	:002000.0 * :05
28B: 0.001775:	:002100.0 * :05
28C: 0.001551:	:002000.0 * :05

08-250075

270365-79

Absolute Error, SN = 4130 (Continued)

28D: 0.001620:		100500 *	1005
28E: 0.001599:		100500 *	1005
28F: 0.001536:		100500 *	1005
290: 0.001558:		100500 *	1005
291: 0.001423:		100500 *	1005
292: 0.001437:		100500 *	1005
293: 0.001255:		100500 *	1005
294: 0.001423:		100500 *	1005
295: 0.001151:		100500 *	1005
296: 0.001336:		100500 *	1005
297: 0.001311:		100500 *	1005
298: 0.001308:		100500 *	1005
299: 0.001125:		100500 *	1005
29A: 0.001060:		100500 *	1005
29B: 0.001134:		100500 *	1005
29C: 0.001209:		100500 *	1005
29D: 0.000856:		100500 *	1005
29E: 0.001095:		100500 *	1005
29F: 0.000790:		100500 *	1005
2A0: 0.000988:		100500 *	1005
2A1: 0.000839:		100500 *	1005
2A2: 0.001122:		100500 *	1005
2A3: 0.000913:		100500 *	1005
2A4: 0.000971:		100500 *	1005
2A5: 0.000710:		100500 *	1005
2A6: 0.000879:		100500 *	1005
2A7: 0.000807:		100500 *	1005
2A8: 0.001102:		100500 *	1005
2A9: 0.000720:		100500 *	1005
2AA: -0.000620:	*	100500 *	1005
2AB: 0.000799:		100500 *	1005
2AC: 0.000991:		100500 *	1005
2AD: 0.000727:		100500 *	1005
2AE: 0.000684:		100500 *	1005
2AF: 0.000683:		100500 *	1005
2B0: 0.000713:		100500 *	1005
2B1: -0.000782:	*	100500 *	1005
2B2: 0.000601:		100500 *	1005
2B3: -0.000704:	*	100500 *	1005
2B4: 0.000647:		100500 *	1005
2B5: -0.000815:	*	100500 *	1005
2B6: -0.000685:	*	100500 *	1005
2B7: -0.000716:	*	100500 *	1005
2B8: 0.000688:		100500 *	1005
2B9: -0.000764:	*	100500 *	1005
2BA: -0.000661:	*	100500 *	1005
2BB: -0.000781:	*	100500 *	1005
2BC: 0.000904:		100500 *	1005
2BD: 0.000707:		100500 *	1005
2BE: 0.000763:		100500 *	1005
2BF: 0.000844:		100500 *	1005
2C0: 0.002248:		100500 *	1005
2C1: 0.001988:		100500 *	1005
2C2: 0.002117:		100500 *	1005
2C3: 0.002005:		100500 *	1005
2C4: 0.002275:		100500 *	1005
2C5: 0.002183:		100500 *	1005
2C6: 0.002092:		100500 *	1005
2C7: 0.002171:		100500 *	1005
2C8: 0.002366:		100500 *	1005

01-00013

270365-80

Absolute Error, SN = 4130 (Continued)

2C9:	0.002105:		:145100.0	:100E
2CA:	0.002047:		:146100.0	:100E
2CB:	0.002142:		:147100.0	:100E
2CC:	0.002308:		:148100.0	:100E
2CD:	0.002226:		:149100.0	:100E
2CE:	0.002106:		:150100.0	:100E
2CF:	0.001931:		:151100.0	:100E
2D0:	0.002298:		:152100.0	:100E
2D1:	0.001963:		:153100.0	:100E
2D2:	0.002106:		:154100.0	:100E
2D3:	0.002014:		:155100.0	:100E
2D4:	0.002136:		:156100.0	:100E
2D5:	0.001849:		:157100.0	:100E
2D6:	0.002152:		:158100.0	:100E
2D7:	0.002205:		:159100.0	:100E
2D8:	0.002087:		:160100.0	:100E
2D9:	0.001866:		:161100.0	:100E
2DA:	0.002304:		:162100.0	:100E
2DB:	0.002234:		:163100.0	:100E
2DC:	0.002308:		:164100.0	:100E
2DD:	0.001769:		:165100.0	:100E
2DE:	0.002155:		:166100.0	:100E
2DF:	0.002034:		:167100.0	:100E
2E0:	0.001801:		:168100.0	:100E
2E1:	0.001788:		:169100.0	:100E
2E2:	0.001813:		:170100.0	:100E
2E3:	0.001724:		:171100.0	:100E
2E4:	0.001537:		:172100.0	:100E
2E5:	0.001622:		:173100.0	:100E
2E6:	0.001797:		:174100.0	:100E
2E7:	0.001799:		:175100.0	:100E
2E8:	0.001720:		:176100.0	:100E
2E9:	0.001537:		:177100.0	:100E
2EA:	0.001715:		:178100.0	:100E
2EB:	0.001385:		:179100.0	:100E
2EC:	0.001687:		:180100.0	:100E
2ED:	0.001464:		:181100.0	:100E
2EE:	0.001508:		:182100.0	:100E
2EF:	0.001373:		:183100.0	:100E
2F0:	0.001488:		:184100.0	:100E
2F1:	0.001379:		:185100.0	:100E
2F2:	0.001508:		:186100.0	:100E
2F3:	0.001325:		:187100.0	:100E
2F4:	0.001385:		:188100.0	:100E
2F5:	0.001225:		:189100.0	:100E
2F6:	0.001381:		:190100.0	:100E
2F7:	0.001301:		:191100.0	:100E
2F8:	0.001168:		:192100.0	:100E
2F9:	0.001136:		:193100.0	:100E
2FA:	0.001032:		:194100.0	:100E
2FB:	0.000957:		:195100.0	:100E
2FC:	0.001102:		:196100.0	:100E
2FD:	0.001088:		:197100.0	:100E
2FE:	0.000999:		:198100.0	:100E
2FF:	0.001571:		:199100.0	:100E
300:	0.001484:		:200100.0	:100E
301:	0.001278:		:201100.0	:100E
302:	0.001463:		:202100.0	:100E
303:	0.001298:		:203100.0	:100E
304:	0.001282:		:204100.0	:100E

6

28-550271

270365-81

Absolute Error, SN = 4130 (Continued)

341:	0.002709:		:018100.0	:01*
342:	0.002828:		:058100.0	:01*
343:	0.002542:		:008500.0	:01*
344:	0.002784:		:000500.0	:00*
345:	0.002719:		:000100.0	:01*
346:	0.002590:		:007100.0	:01*
347:	0.002871:		:000100.0	:00*
348:	0.003014:		:007100.0	:00*
349:	0.003003:		:000100.0	:00*
34A:	0.002773:		:000100.0	:00*
34B:	0.002744:		:000100.0	:00*
34C:	0.003031:		:007100.0	:00*
34D:	0.002672:		:000100.0	:00*
34E:	0.002854:		:000100.0	:00*
34F:	0.002906:		:000100.0	:00*
350:	0.002960:		:000100.0	:00*
351:	0.002742:		:000100.0	:00*
352:	0.002836:		:000100.0	:00*
353:	0.002754:		:000100.0	:00*
354:	0.003072:		:000100.0	:00*
355:	0.002821:		:000100.0	:00*
356:	0.003011:		:000100.0	:00*
357:	0.003037:		:000100.0	:00*
358:	0.002763:		:000100.0	:00*
359:	0.002649:		:000100.0	:00*
35A:	0.002595:		:000100.0	:00*
35B:	0.002773:		:000100.0	:00*
35C:	0.002793:		:000100.0	:00*
35D:	0.002479:		:000100.0	:00*
35E:	0.002709:		:000100.0	:00*
35F:	0.002716:		:000100.0	:00*
360:	0.002505:		:000100.0	:00*
361:	0.002437:		:000100.0	:00*
362:	0.002451:		:000100.0	:00*
363:	0.002320:		:000100.0	:00*
364:	0.002448:		:000100.0	:00*
365:	0.002264:		:000100.0	:00*
366:	0.002375:		:000100.0	:00*
367:	0.002312:		:000100.0	:00*
368:	0.002421:		:000100.0	:00*
369:	0.002251:		:000100.0	:00*
36A:	0.002330:		:000100.0	:00*
36B:	0.002272:		:000100.0	:00*
36C:	0.002269:		:000100.0	:00*
36D:	0.001925:		:000100.0	:00*
36E:	0.002158:		:000100.0	:00*
36F:	0.002229:		:000100.0	:00*
370:	0.002246:		:000100.0	:00*
371:	0.001929:		:000100.0	:00*
372:	0.002095:		:000100.0	:00*
373:	0.002046:		:000100.0	:00*
374:	0.002085:		:000100.0	:00*
375:	0.001876:		:000100.0	:00*
376:	0.001926:		:000100.0	:00*
377:	0.002039:		:000100.0	:00*
378:	0.001967:		:000100.0	:00*
379:	0.001932:		:000100.0	:00*
37A:	0.002019:		:000100.0	:00*
37B:	0.001950:		:000100.0	:00*
37C:	0.001922:		:000100.0	:00*

48-230172

270365-83

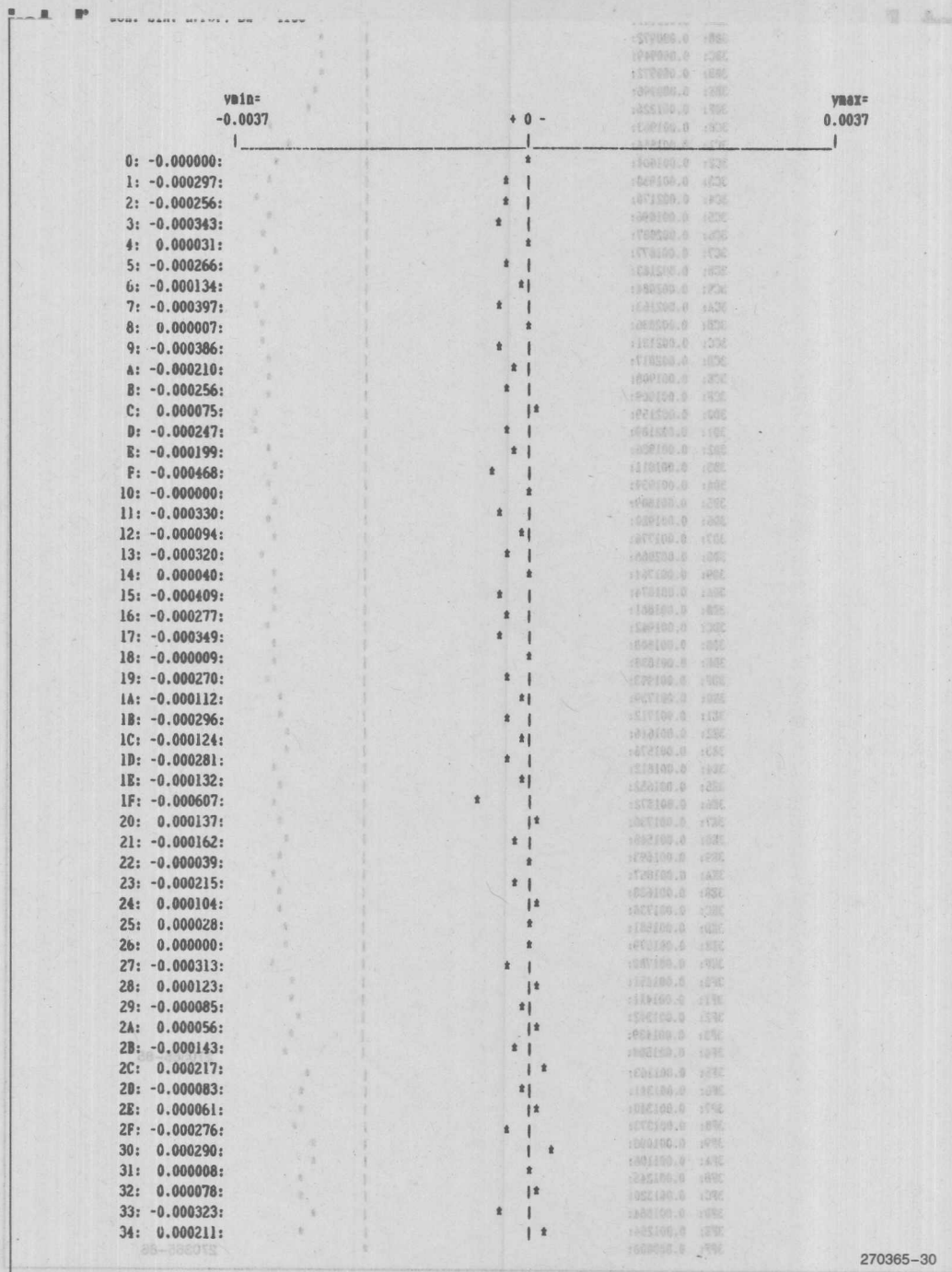
37D: 0.001815:	1804500.0	1780
37E: 0.001689:	1808500.0	1580
37F: 0.002200:	1812500.0	1580
380: 0.002064:	1816500.0	1580
381: 0.001764:	1820500.0	1580
382: 0.001910:	1824500.0	1580
383: 0.001945:	1828500.0	1580
384: 0.001913:	1832500.0	1580
385: 0.001866:	1836500.0	1580
386: 0.001889:	1840500.0	1580
387: 0.001800:	1844500.0	1580
388: 0.001779:	1848500.0	1580
389: 0.001454:	1852500.0	1580
38A: 0.001584:	1856500.0	1580
38B: 0.001477:	1860500.0	1580
38C: 0.001469:	1864500.0	1580
38D: 0.001268:	1868500.0	1580
38E: 0.001562:	1872500.0	1580
38F: 0.001268:	1876500.0	1580
390: 0.001568:	1880500.0	1580
391: 0.000946:	1884500.0	1580
392: 0.001423:	1888500.0	1580
393: 0.001232:	1892500.0	1580
394: 0.001499:	1896500.0	1580
395: 0.001255:	1900500.0	1580
396: 0.001087:	1904500.0	1580
397: 0.001265:	1908500.0	1580
398: 0.001421:	1912500.0	1580
399: 0.001169:	1916500.0	1580
39A: 0.001269:	1920500.0	1580
39B: 0.001245:	1924500.0	1580
39C: 0.001440:	1928500.0	1580
39D: 0.001153:	1932500.0	1580
39E: 0.001402:	1936500.0	1580
39F: 0.001260:	1940500.0	1580
3A0: 0.001363:	1944500.0	1580
3A1: 0.001145:	1948500.0	1580
3A2: 0.001221:	1952500.0	1580
3A3: 0.001155:	1956500.0	1580
3A4: 0.001452:	1960500.0	1580
3A5: 0.001302:	1964500.0	1580
3A6: 0.001138:	1968500.0	1580
3A7: 0.001079:	1972500.0	1580
3A8: 0.001378:	1976500.0	1580
3A9: 0.001043:	1980500.0	1580
3AA: 0.001145:	1984500.0	1580
3AB: 0.001207:	1988500.0	1580
3AC: 0.001161:	1992500.0	1580
3AD: 0.001133:	1996500.0	1580
3AE: 0.001137:	2000500.0	1580
3AF: 0.001175:	2004500.0	1580
3B0: 0.001159:	2008500.0	1580
3B1: 0.000747:	2012500.0	1580
3B2: 0.000927:	2016500.0	1580
3B3: 0.000883:	2020500.0	1580
3B4: 0.001127:	2024500.0	1580
3B5: 0.000784:	2028500.0	1580
3B6: 0.001002:	2032500.0	1580
3B7: 0.001058:	2036500.0	1580
3B8: 0.000907:	2040500.0	1580

28-282075

270365-84

Absolute Error, SN = 4130 (Continued)

6-253



270365-30

35: -0.000065:	* 000000.0 111
36: 0.000165:	*000000.0 151
37: -0.000106:	* 000000.0 151
38: 0.000409:	*000000.0 151
39: 0.000186:	*000000.0 121
3A: 0.000398:	*000000.0 101
3B: 0.000166:	*000000.0 111
3C: 0.000368:	*000000.0 101
3D: 0.000287:	*000000.0 101
3E: 0.000513:	000000.0 101
3F: -0.000275:	* 000000.0 101
40: 0.002147:	000000.0 101
41: 0.001651:	000000.0 101
42: 0.001983:	000000.0 101
43: 0.001784:	000000.0 101
44: 0.001994:	000000.0 101
45: 0.001478:	000000.0 101
46: 0.001922:	000000.0 101
47: 0.001617:	000000.0 101
48: 0.001910:	000000.0 101
49: 0.001616:	000000.0 101
4A: 0.001833:	000000.0 101
4B: 0.001621:	000000.0 101
4C: 0.001872:	000000.0 101
4D: 0.001585:	000000.0 101
4E: 0.001771:	000000.0 101
4F: 0.001392:	000000.0 101
50: 0.001900:	000000.0 101
51: 0.001682:	000000.0 101
52: 0.001571:	000000.0 101
53: 0.001498:	000000.0 101
54: 0.001755:	000000.0 101
55: 0.001485:	000000.0 101
56: 0.001594:	000000.0 101
57: 0.001455:	000000.0 101
58: 0.001641:	000000.0 101
59: 0.001382:	000000.0 101
5A: 0.001604:	000000.0 101
5B: 0.001489:	000000.0 101
5C: 0.001650:	000000.0 101
5D: 0.001323:	000000.0 101
5E: 0.001536:	000000.0 101
5F: 0.000952:	000000.0 101
60: 0.001437:	000000.0 101
61: 0.001163:	000000.0 101
62: 0.001365:	000000.0 101
63: 0.001156:	000000.0 101
64: 0.001275:	000000.0 101
65: 0.000971:	000000.0 101
66: 0.001141:	000000.0 101
67: 0.000923:	000000.0 101
68: 0.000980:	000000.0 101
69: 0.000803:	000000.0 101
6A: 0.000797:	000000.0 101
6B: 0.000806:	000000.0 101
6C: 0.000928:	000000.0 101
6D: 0.000589:	000000.0 101
6E: 0.000793:	000000.0 101
6F: 0.000592:	000000.0 101
70: 0.000798:	000000.0 101

38-38003

270365-34

Non. Lin. Error, SN = 4130 (Continued)

71: 0.000442:	:25000.0 :02
72: 0.000576:	:75000.0 :02
73: 0.000537:	:00000.0 :02
74: 0.000616:	:00000.0 :02
75: 0.000216:	:00000.0 :02
76: 0.000493:	:00000.0 :02
77: 0.000393:	:00000.0 :02
78: 0.000330:	:00000.0 :02
79: 0.000111:	:00000.0 :02
7A: 0.000216:	:00000.0 :02
7B: 0.000158:	:00000.0 :02
7C: 0.000148:	:00000.0 :02
7D: -0.000060:	* :00000.0 :02
7E: 0.000081:	:00000.0 :02
7F: -0.000601:	* :00000.0 :02
80: 0.000691:	:00000.0 :02
81: 0.000447:	:00000.0 :02
82: 0.000588:	:00000.0 :02
83: 0.000434:	:00000.0 :02
84: 0.000566:	:00000.0 :02
85: 0.000265:	:00000.0 :02
86: 0.000397:	:00000.0 :02
87: 0.000157:	:00000.0 :02
88: 0.000396:	:00000.0 :02
89: -0.000196:	* :00000.0 :02
8A: 0.000339:	:00000.0 :02
8B: -0.000021:	* :00000.0 :02
8C: 0.000166:	:00000.0 :02
8D: -0.000104:	* :00000.0 :02
8E: -0.000163:	* :00000.0 :02
8F: -0.000285:	* :00000.0 :02
90: 0.000089:	:00000.0 :02
91: -0.000055:	* :00000.0 :02
92: 0.000057:	:00000.0 :02
93: -0.000129:	* :00000.0 :02
94: 0.000025:	:00000.0 :02
95: -0.000194:	* :00000.0 :02
96: -0.000048:	* :00000.0 :02
97: -0.000255:	* :00000.0 :02
98: -0.000119:	* :00000.0 :02
99: -0.000445:	* :00000.0 :02
9A: -0.000214:	* :00000.0 :02
9B: -0.000376:	* :00000.0 :02
9C: -0.000305:	* :00000.0 :02
9D: -0.000650:	* :00000.0 :02
9E: -0.000467:	* :00000.0 :02
9F: -0.000967:	* :00000.0 :02
A0: -0.000481:	* :00000.0 :02
A1: -0.000830:	* :00000.0 :02
A2: -0.000416:	* :00000.0 :02
A3: -0.000790:	* :00000.0 :02
A4: -0.000574:	* :00000.0 :02
A5: -0.000848:	* :00000.0 :02
A6: -0.000709:	* :00000.0 :02
A7: -0.000898:	* :00000.0 :02
A8: -0.000774:	* :00000.0 :02
A9: -0.000892:	* :00000.0 :02
AA: -0.000768:	* :00000.0 :02
AB: -0.000911:	* :00000.0 :02
AC: -0.000824:	* :00000.0 :02

NE-2800'S

270365-35

Non. Lin. Error, SN = 4130 (Continued)

6

6-257

E9: -0.000694:	100000.0	100
EA: -0.000557:	100000.0	100
EB: -0.000709:	100000.0	100
EC: -0.000540:	100000.0	100
ED: -0.000752:	100000.0	100
EE: -0.000557:	100000.0	100
EF: -0.000728:	100000.0	100
EO: -0.000612:	100000.0	100
FI: -0.000989:	100000.0	100
F2: -0.000780:	100000.0	100
F3: -0.000947:	100000.0	100
F4: -0.000868:	100000.0	100
F5: -0.001156:	100000.0	100
F6: -0.000807:	100000.0	100
F7: -0.001038:	100000.0	100
F8: -0.001200:	100000.0	100
F9: -0.001222:	100000.0	100
FA: -0.000956:	100000.0	100
FB: -0.001087:	100000.0	100
FC: -0.000919:	100000.0	100
FD: -0.001063:	100000.0	100
FE: -0.000977:	100000.0	100
FF: -0.001857:	100000.0	100
100: -0.000509:	100000.0	100
101: -0.000843:	100000.0	100
102: -0.000606:	100000.0	100
103: -0.000828:	100000.0	100
104: -0.000859:	100000.0	100
105: -0.000982:	100000.0	100
106: -0.000973:	100000.0	100
107: -0.001042:	100000.0	100
108: -0.000775:	100000.0	100
109: -0.001040:	100000.0	100
10A: -0.000904:	100000.0	100
10B: -0.000982:	100000.0	100
10C: -0.000912:	100000.0	100
10D: -0.001173:	100000.0	100
10E: -0.001030:	100000.0	100
10F: -0.001382:	100000.0	100
110: -0.001240:	100000.0	100
111: -0.001386:	100000.0	100
112: -0.001093:	100000.0	100
113: -0.001244:	100000.0	100
114: -0.001240:	100000.0	100
115: -0.001503:	100000.0	100
116: -0.001328:	100000.0	100
117: -0.001480:	100000.0	100
118: -0.001401:	100000.0	100
119: -0.001521:	100000.0	100
11A: -0.001494:	100000.0	100
11B: -0.001614:	100000.0	100
11C: -0.001565:	100000.0	100
11D: -0.001814:	100000.0	100
11E: -0.001621:	100000.0	100
11F: -0.002076:	100000.0	100
120: -0.001541:	100000.0	100
121: -0.001841:	100000.0	100
122: -0.001701:	100000.0	100
123: -0.001790:	100000.0	100
124: -0.001644:	100000.0	100

270365-37

Non. Lin. Error, SN = 4130 (Continued)

125: -0.002140:	*	125100.0-100
126: -0.001946:	*	126100.0-100
127: -0.001975:	*	127100.0-100
128: -0.001772:	*	128100.0-100
129: -0.001967:	*	129100.0-100
12A: -0.001774:	*	12A100.0-100
12B: -0.001888:	*	12B100.0-100
12C: -0.001948:	*	12C100.0-100
12D: -0.002097:	*	12D100.0-100
12E: -0.002048:	*	12E100.0-100
12F: -0.002148:	*	12F100.0-100
130: -0.001980:	*	130100.0-100
131: -0.002243:	*	131100.0-100
132: -0.002054:	*	132100.0-100
133: -0.002233:	*	133100.0-100
134: -0.002039:	*	134100.0-100
135: -0.002342:	*	135100.0-100
136: -0.002083:	*	136100.0-100
137: -0.002333:	*	137100.0-100
138: -0.001896:	*	138100.0-100
139: -0.002125:	*	139100.0-100
13A: -0.002177:	*	13A100.0-100
13B: -0.002168:	*	13B100.0-100
13C: -0.001897:	*	13C100.0-100
13D: -0.002142:	*	13D100.0-100
13E: -0.002018:	*	13E100.0-100
13F: -0.002490:	*	13F100.0-100
140: -0.000666:	*	140100.0-100
141: -0.000700:	*	141100.0-100
142: -0.000560:	*	142100.0-100
143: -0.000692:	*	143100.0-100
144: -0.000493:	*	144100.0-100
145: -0.000724:	*	145100.0-100
146: -0.000607:	*	146100.0-100
147: -0.000659:	*	147100.0-100
148: -0.000441:	*	148100.0-100
149: -0.000712:	*	149100.0-100
14A: -0.000578:	*	14A100.0-100
14B: -0.000517:	*	14B100.0-100
14C: -0.000527:	*	14C100.0-100
14D: -0.000753:	*	14D100.0-100
14E: -0.000650:	*	14E100.0-100
14F: -0.000857:	*	14F100.0-100
150: -0.000688:	*	150100.0-100
151: -0.000880:	*	151100.0-100
152: -0.000746:	*	152100.0-100
153: -0.000917:	*	153100.0-100
154: -0.000747:	*	154100.0-100
155: -0.000986:	*	155100.0-100
156: -0.000929:	*	156100.0-100
157: -0.000931:	*	157100.0-100
158: -0.000731:	*	158100.0-100
159: -0.001008:	*	159100.0-100
15A: -0.000898:	*	15A100.0-100
15B: -0.000972:	*	15B100.0-100
15C: -0.000867:	*	15C100.0-100
15D: -0.001075:	*	15D100.0-100
15E: -0.000914:	*	15E100.0-100
15F: -0.001140:	*	15F100.0-100
160: -0.001002:	*	160100.0-100

9C-280075

270365-38

Non. Lin. Error, SN = 4130 (Continued)

161: -0.001273:	* :001500.0- :001
162: -0.001057:	* :0100.0- :001
163: -0.001275:	* :07100.0- :001
164: -0.001048:	* :07100.0- :001
165: -0.001502:	* :00100.0- :001
166: -0.001155:	* :07100.0- :001
167: -0.001274:	* :00100.0- :001
168: -0.001100:	* :00100.0- :001
169: -0.001259:	* :00100.0- :001
16A: -0.000968:	* :00100.0- :001
16B: -0.001205:	* :00100.0- :001
16C: -0.001170:	* :00100.0- :001
16D: -0.001449:	* :00100.0- :001
16E: -0.001375:	* :00100.0- :001
16F: -0.001347:	* :00100.0- :001
170: -0.001278:	* :00100.0- :001
171: -0.001557:	* :00100.0- :001
172: -0.001365:	* :00100.0- :001
173: -0.001430:	* :00100.0- :001
174: -0.001328:	* :00100.0- :001
175: -0.001520:	* :00100.0- :001
176: -0.001455:	* :00100.0- :001
177: -0.001480:	* :00100.0- :001
178: -0.001315:	* :00100.0- :001
179: -0.001617:	* :00100.0- :001
17A: -0.001338:	* :00100.0- :001
17B: -0.001484:	* :00100.0- :001
17C: -0.001486:	* :00100.0- :001
17D: -0.001679:	* :00100.0- :001
17E: -0.001500:	* :00100.0- :001
17F: -0.001611:	* :00100.0- :001
180: -0.001098:	* :00100.0- :001
181: -0.001379:	* :00100.0- :001
182: -0.001306:	* :00100.0- :001
183: -0.001366:	* :00100.0- :001
184: -0.001230:	* :00100.0- :001
185: -0.001478:	* :00100.0- :001
186: -0.001377:	* :00100.0- :001
187: -0.001480:	* :00100.0- :001
188: -0.001309:	* :00100.0- :001
189: -0.001553:	* :00100.0- :001
18A: -0.001378:	* :00100.0- :001
18B: -0.001493:	* :00100.0- :001
18C: -0.001353:	* :00100.0- :001
18D: -0.001682:	* :00100.0- :001
18E: -0.001502:	* :00100.0- :001
18F: -0.001678:	* :00100.0- :001
190: -0.001229:	* :00100.0- :001
191: -0.001624:	* :00100.0- :001
192: -0.001671:	* :00100.0- :001
193: -0.001674:	* :00100.0- :001
194: -0.001672:	* :00100.0- :001
195: -0.001841:	* :00100.0- :001
196: -0.001669:	* :00100.0- :001
197: -0.002024:	* :00100.0- :001
198: -0.001466:	* :00100.0- :001
199: -0.001871:	* :00100.0- :001
19A: -0.001688:	* :00100.0- :001
19B: -0.001782:	* :00100.0- :001
19C: -0.001516:	* :00100.0- :001

82-880075

270365-39

Non. Lin. Error, SN = 4130 (Continued)

19D: -0.001845:	*	102000.0	101
19E: -0.001864:	*	102100.0	101
19F: -0.001909:	*	102200.0	101
1A0: -0.001698:	*	102300.0	101
1A1: -0.001943:	*	102400.0	101
1A2: -0.001803:	*	102500.0	101
1A3: -0.001894:	*	102600.0	101
1A4: -0.001936:	*	102700.0	101
1A5: -0.002146:	*	102800.0	101
1A6: -0.001983:	*	102900.0	101
1A7: -0.002040:	*	103000.0	101
1A8: -0.001877:	*	103100.0	101
1A9: -0.002035:	*	103200.0	101
1AA: -0.001921:	*	103300.0	101
1AB: -0.001896:	*	103400.0	101
1AC: -0.001867:	*	103500.0	101
1AD: -0.002108:	*	103600.0	101
1AE: -0.001997:	*	103700.0	101
1AF: -0.001807:	*	103800.0	101
1B0: -0.001822:	*	103900.0	101
1B1: -0.002051:	*	104000.0	101
1B2: -0.001916:	*	104100.0	101
1B3: -0.001991:	*	104200.0	101
1B4: -0.001973:	*	104300.0	101
1B5: -0.002108:	*	104400.0	101
1B6: -0.002067:	*	104500.0	101
1B7: -0.002013:	*	104600.0	101
1B8: -0.002053:	*	104700.0	101
1B9: -0.002113:	*	104800.0	101
1BA: -0.001913:	*	104900.0	101
1BB: -0.001950:	*	105000.0	101
1BC: -0.001974:	*	105100.0	101
1BD: -0.002144:	*	105200.0	101
1BE: -0.002055:	*	105300.0	101
1BF: -0.002041:	*	105400.0	101
1C0: -0.000629:	*	105500.0	101
1C1: -0.000747:	*	105600.0	101
1C2: -0.000569:	*	105700.0	101
1C3: -0.000701:	*	105800.0	101
1C4: -0.000497:	*	105900.0	101
1C5: -0.000746:	*	106000.0	101
1C6: -0.000533:	*	106100.0	101
1C7: -0.000677:	*	106200.0	101
1C8: -0.000419:	*	106300.0	101
1C9: -0.000728:	*	106400.0	101
1CA: -0.000699:	*	106500.0	101
1CB: -0.000643:	*	106600.0	101
1CC: -0.000509:	*	106700.0	101
1CD: -0.000759:	*	106800.0	101
1CE: -0.000676:	*	106900.0	101
1CF: -0.000678:	*	107000.0	101
1D0: -0.000508:	*	107100.0	101
1D1: -0.000778:	*	107200.0	101
1D2: -0.000675:	*	107300.0	101
1D3: -0.000726:	*	107400.0	101
1D4: -0.000558:	*	107500.0	101
1D5: -0.000768:	*	107600.0	101
1D6: -0.000760:	*	107700.0	101
1D7: -0.000645:	*	107800.0	101
1D8: -0.000580:	*	107900.0	101

27-58005

270365-40

Non. Lin. Error, SN = 4130 (Continued)

1D9: -0.000836:	*	:000000.0-:001
1DA: -0.000750:	*	:000000.0-:001
1DB: -0.000758:	*	:000000.0-:001
1DC: -0.000715:	*	:000000.0-:001
1DD: -0.000816:	*	:000000.0-:001
1DE: -0.000701:	*	:000000.0-:001
1DF: -0.000714:	*	:000000.0-:001
1E0: -0.000663:	*	:000000.0-:001
1E1: -0.001060:	*	:000000.0-:001
1E2: -0.000828:	*	:000000.0-:001
1E3: -0.001107:	*	:000000.0-:001
1E4: -0.000802:	*	:000000.0-:001
1E5: -0.001037:	*	:000000.0-:001
1E6: -0.000803:	*	:000000.0-:001
1E7: -0.000843:	*	:000000.0-:001
1E8: -0.000702:	*	:000000.0-:001
1E9: -0.001156:	*	:000000.0-:001
1EA: -0.000861:	*	:000000.0-:001
1EB: -0.000965:	*	:000000.0-:001
1EC: -0.000933:	*	:000000.0-:001
1ED: -0.001142:	*	:000000.0-:001
1EE: -0.001205:	*	:000000.0-:001
1EF: -0.000995:	*	:000000.0-:001
1F0: -0.000954:	*	:000000.0-:001
1F1: -0.001179:	*	:000000.0-:001
1F2: -0.001084:	*	:000000.0-:001
1F3: -0.001061:	*	:000000.0-:001
1F4: -0.001035:	*	:000000.0-:001
1F5: -0.001099:	*	:000000.0-:001
1F6: -0.001111:	*	:000000.0-:001
1F7: -0.000960:	*	:000000.0-:001
1F8: -0.000991:	*	:000000.0-:001
1F9: -0.001178:	*	:000000.0-:001
1FA: -0.001061:	*	:000000.0-:001
1FB: -0.001099:	*	:000000.0-:001
1FC: -0.001026:	*	:000000.0-:001
1FD: -0.001248:	*	:000000.0-:001
1FE: -0.001157:	*	:000000.0-:001
1FF: -0.001828:	*	:000000.0-:001
200: -0.001360:	*	:000000.0-:001
201: -0.001583:	*	:000000.0-:001
202: -0.001386:	*	:000000.0-:001
203: -0.001636:	*	:000000.0-:001
204: -0.001536:	*	:000000.0-:001
205: -0.001584:	*	:000000.0-:001
206: -0.001514:	*	:000000.0-:001
207: -0.001703:	*	:000000.0-:001
208: -0.001714:	*	:000000.0-:001
209: -0.002021:	*	:000000.0-:001
20A: -0.001592:	*	:000000.0-:001
20B: -0.001799:	*	:000000.0-:001
20C: -0.001884:	*	:000000.0-:001
20D: -0.001994:	*	:000000.0-:001
20E: -0.002028:	*	:000000.0-:001
20F: -0.002225:	*	:000000.0-:001
210: -0.001805:	*	:000000.0-:001
211: -0.002137:	*	:000000.0-:001
212: -0.001994:	*	:000000.0-:001
213: -0.002071:	*	:000000.0-:001
214: -0.001795:	*	:000000.0-:001

04-280075

270365-41

Non. Lin. Error, SN = 4130 (Continued)

215: -0.002104:	*	1000100.0- :100
216: -0.001945:	*	1000100.0- :100
217: -0.002001:	*	1000100.0- :100
218: -0.001974:	*	1000100.0- :100
219: -0.002175:	*	1000100.0- :100
21A: -0.002187:	*	1000100.0- :100
21B: -0.002104:	*	1000100.0- :100
21C: -0.001725:	*	1000100.0- :100
21D: -0.002212:	*	1000100.0- :100
21E: -0.002012:	*	1000100.0- :100
21F: -0.002564:	*	1000100.0- :100
220: -0.002027:	*	1000100.0- :100
221: -0.002294:	*	1000100.0- :100
222: -0.002127:	*	1000100.0- :100
223: -0.002269:	*	1000100.0- :100
224: -0.002157:	*	1000100.0- :100
225: -0.002308:	*	1000100.0- :100
226: -0.002268:	*	1000100.0- :100
227: -0.002372:	*	1000100.0- :100
228: -0.002039:	*	1000100.0- :100
229: -0.002344:	*	1000100.0- :100
22A: -0.002218:	*	1000100.0- :100
22B: -0.002244:	*	1000100.0- :100
22C: -0.002179:	*	1000100.0- :100
22D: -0.002361:	*	1000100.0- :100
22E: -0.002101:	*	1000100.0- :100
22F: -0.002463:	*	1000100.0- :100
230: -0.002088:	*	1000100.0- :100
231: -0.002333:	*	1000100.0- :100
232: -0.002052:	*	1000100.0- :100
233: -0.002328:	*	1000100.0- :100
234: -0.002104:	*	1000100.0- :100
235: -0.002278:	*	1000100.0- :100
236: -0.002357:	*	1000100.0- :100
237: -0.002259:	*	1000100.0- :100
238: -0.002078:	*	1000100.0- :100
239: -0.002559:	*	1000100.0- :100
23A: -0.002199:	*	1000100.0- :100
23B: -0.002343:	*	1000100.0- :100
23C: -0.002181:	*	1000100.0- :100
23D: -0.002369:	*	1000100.0- :100
23E: -0.002265:	*	1000100.0- :100
23F: -0.002833:	*	1000100.0- :100
240: -0.001187:	*	1000100.0- :100
241: -0.001357:	*	1000100.0- :100
242: -0.001130:	*	1000100.0- :100
243: -0.001301:	*	1000100.0- :100
244: -0.001167:	*	1000100.0- :100
245: -0.001462:	*	1000100.0- :100
246: -0.001157:	*	1000100.0- :100
247: -0.001412:	*	1000100.0- :100
248: -0.001224:	*	1000100.0- :100
249: -0.001278:	*	1000100.0- :100
24A: -0.001187:	*	1000100.0- :100
24B: -0.001278:	*	1000100.0- :100
24C: -0.001023:	*	1000100.0- :100
24D: -0.001256:	*	1000100.0- :100
24E: -0.001147:	*	1000100.0- :100
24F: -0.001204:	*	1000100.0- :100
250: -0.000930:	*	1000100.0- :100

ED-000075

270365-42

Non. Lin. Error, SN = 4130 (Continued)

251: -0.001283:	* 101500.0- 1015
252: -0.001088:	* 101600.0- 1015
253: -0.001281:	* 101700.0- 1015
254: -0.000930:	* 101800.0- 1015
255: -0.001188:	* 101900.0- 1015
256: -0.001039:	* 102000.0- 1015
257: -0.001147:	* 102100.0- 1015
258: -0.001096:	* 102200.0- 1015
259: -0.001217:	* 102300.0- 1015
25A: -0.001302:	* 102400.0- 1015
25B: -0.001214:	* 102500.0- 1015
25C: -0.001084:	* 102600.0- 1015
25D: -0.001276:	* 102700.0- 1015
25E: -0.001273:	* 102800.0- 1015
25F: -0.001343:	* 102900.0- 1015
260: -0.001229:	* 103000.0- 1015
261: -0.001509:	* 103100.0- 1015
262: -0.001297:	* 103200.0- 1015
263: -0.001426:	* 103300.0- 1015
264: -0.001318:	* 103400.0- 1015
265: -0.001517:	* 103500.0- 1015
266: -0.001282:	* 103600.0- 1015
267: -0.001485:	* 103700.0- 1015
268: -0.001357:	* 103800.0- 1015
269: -0.001616:	* 103900.0- 1015
26A: -0.001509:	* 104000.0- 1015
26B: -0.001558:	* 104100.0- 1015
26C: -0.001582:	* 104200.0- 1015
26D: -0.001748:	* 104300.0- 1015
26E: -0.001524:	* 104400.0- 1015
26F: -0.001692:	* 104500.0- 1015
270: -0.001589:	* 104600.0- 1015
271: -0.001786:	* 104700.0- 1015
272: -0.001708:	* 104800.0- 1015
273: -0.001751:	* 104900.0- 1015
274: -0.001716:	* 105000.0- 1015
275: -0.001988:	* 105100.0- 1015
276: -0.001813:	* 105200.0- 1015
277: -0.001816:	* 105300.0- 1015
278: -0.001943:	* 105400.0- 1015
279: -0.002046:	* 105500.0- 1015
27A: -0.001936:	* 105600.0- 1015
27B: -0.002000:	* 105700.0- 1015
27C: -0.001783:	* 105800.0- 1015
27D: -0.002248:	* 105900.0- 1015
27E: -0.001869:	* 106000.0- 1015
27F: -0.002131:	* 106100.0- 1015
280: -0.001496:	* 106200.0- 1015
281: -0.001510:	* 106300.0- 1015
282: -0.001316:	* 106400.0- 1015
283: -0.001792:	* 106500.0- 1015
284: -0.001318:	* 106600.0- 1015
285: -0.001615:	* 106700.0- 1015
286: -0.001465:	* 106800.0- 1015
287: -0.001632:	* 106900.0- 1015
288: -0.001643:	* 107000.0- 1015
289: -0.001730:	* 107100.0- 1015
28A: -0.001629:	* 107200.0- 1015
28B: -0.001698:	* 107300.0- 1015
28C: -0.001823:	* 107400.0- 1015

31-280712

270365-43

Non. Lin. Error, SN = 4130 (Continued)

28D: -0.001855:	*	028100.0: 171
28E: -0.001778:	*	028100.0: 172
28F: -0.001942:	*	028100.0: 173
290: -0.001871:	*	028100.0: 174
291: -0.002008:	*	028100.0: 175
292: -0.001945:	*	028100.0: 176
293: -0.002128:	*	028100.0: 177
294: -0.001962:	*	028100.0: 178
295: -0.002235:	*	028100.0: 179
296: -0.002101:	*	028100.0: 180
297: -0.002178:	*	028100.0: 181
298: -0.002132:	*	028100.0: 182
299: -0.002366:	*	028100.0: 183
29A: -0.002433:	*	028100.0: 184
29B: -0.002360:	*	028100.0: 185
29C: -0.002236:	*	028100.0: 186
29D: -0.002541:	*	028100.0: 187
29E: -0.002403:	*	028100.0: 188
29F: -0.002609:	*	028100.0: 189
2A0: -0.002413:	*	028100.0: 190
2A1: -0.002563:	*	028100.0: 191
2A2: -0.002381:	*	028100.0: 192
2A3: -0.002542:	*	028100.0: 193
2A4: -0.002435:	*	028100.0: 194
2A5: -0.002697:	*	028100.0: 195
2A6: -0.002530:	*	028100.0: 196
2A7: -0.002653:	*	028100.0: 197
2A8: -0.002459:	*	028100.0: 198
2A9: -0.002742:	*	028100.0: 199
2AA: -0.002860:	*	028100.0: 200
2AB: -0.002666:	*	028100.0: 201
2AC: -0.002525:	*	028100.0: 202
2AD: -0.002741:	*	028100.0: 203
2AE: -0.002785:	*	028100.0: 204
2AF: -0.002737:	*	028100.0: 205
2B0: -0.002709:	*	028100.0: 206
2B1: -0.003031:	*	028100.0: 207
2B2: -0.002823:	*	028100.0: 208
2B3: -0.002906:	*	028100.0: 209
2B4: -0.002780:	*	028100.0: 210
2B5: -0.003019:	*	028100.0: 211
2B6: -0.002941:	*	028100.0: 212
2B7: -0.002923:	*	028100.0: 213
2B8: -0.002794:	*	028100.0: 214
2B9: -0.002973:	*	028100.0: 215
2BA: -0.002872:	*	028100.0: 216
2BB: -0.002943:	*	028100.0: 217
2BC: -0.002584:	*	028100.0: 218
2BD: -0.002832:	*	028100.0: 219
2BE: -0.002777:	*	028100.0: 220
2BF: -0.002698:	*	028100.0: 221
2C0: -0.001295:	*	028100.0: 222
2C1: -0.001557:	*	028100.0: 223
2C2: -0.001429:	*	028100.0: 224
2C3: -0.001542:	*	028100.0: 225
2C4: -0.001274:	*	028100.0: 226
2C5: -0.001417:	*	028100.0: 227
2C6: -0.001409:	*	028100.0: 228
2C7: -0.001382:	*	028100.0: 229
2C8: -0.001138:	*	028100.0: 230

28-28C075

270365-44

Non. Lin. Error, SN = 4130 (Continued)

2C9: -0.001450:	‡	1285100.0- :E19
2CA: -0.001409:	‡	1287100.0- :E18
2CB: -0.001366:	‡	1289100.0- :E17
2CC: -0.001201:	‡	1291100.0- :E16
2CD: -0.001385:	‡	1293100.0- :E15
2CE: -0.001406:	‡	1295100.0- :E14
2CF: -0.001532:	‡	1297100.0- :E13
2D0: -0.001166:	‡	1299100.0- :E12
2D1: -0.001503:	‡	1301100.0- :E11
2D2: -0.001411:	‡	1303100.0- :E10
2D3: -0.001554:	‡	1305100.0- :E9
2D4: -0.001334:	‡	1307100.0- :E8
2D5: -0.001622:	‡	1309100.0- :E7
2D6: -0.001370:	‡	1311100.0- :E6
2D7: -0.001369:	‡	1313100.0- :E5
2D8: -0.001438:	‡	1315100.0- :E4
2D9: -0.001660:	‡	1317100.0- :E3
2DA: -0.001324:	‡	1319100.0- :E2
2DB: -0.001395:	‡	1321100.0- :E1
2DC: -0.001273:	‡	1323100.0- :E0
2DD: -0.001813:	‡	1325100.0- :E0
2DE: -0.001428:	‡	1327100.0- :E0
2DF: -0.001550:	‡	1329100.0- :E0
2E0: -0.001605:	‡	1331100.0- :E0
2E1: -0.001799:	‡	1333100.0- :E0
2E2: -0.001725:	‡	1335100.0- :E0
2E3: -0.001766:	‡	1337100.0- :E0
2E4: -0.001954:	‡	1339100.0- :E0
2E5: -0.001920:	‡	1341100.0- :E0
2E6: -0.001747:	‡	1343100.0- :E0
2E7: -0.001796:	‡	1345100.0- :E0
2E8: -0.001776:	‡	1347100.0- :E0
2E9: -0.002011:	‡	1349100.0- :E0
2EA: -0.001834:	‡	1351100.0- :E0
2EB: -0.002115:	‡	1353100.0- :E0
2EC: -0.001915:	‡	1355100.0- :E0
2ED: -0.002089:	‡	1357100.0- :E0
2EE: -0.002046:	‡	1359100.0- :E0
2EF: -0.002132:	‡	1361100.0- :E0
2F0: -0.002069:	‡	1363100.0- :E0
2F1: -0.002229:	‡	1365100.0- :E0
2F2: -0.002101:	‡	1367100.0- :E0
2F3: -0.002236:	‡	1369100.0- :E0
2F4: -0.002177:	‡	1371100.0- :E0
2F5: -0.002388:	‡	1373100.0- :E0
2F6: -0.002284:	‡	1375100.0- :E0
2F7: -0.002315:	‡	1377100.0- :E0
2F8: -0.002449:	‡	1379100.0- :E0
2F9: -0.002533:	‡	1381100.0- :E0
2FA: -0.002538:	‡	1383100.0- :E0
2FB: -0.002564:	‡	1385100.0- :E0
2FC: -0.002471:	‡	1387100.0- :E0
2FD: -0.002486:	‡	1389100.0- :E0
2FE: -0.002576:	‡	1391100.0- :E0
2FF: -0.002006:	‡	1393100.0- :E0
300: -0.001994:	‡	1395100.0- :E0
301: -0.002301:	‡	1397100.0- :E0
302: -0.002168:	‡	1399100.0- :E0
303: -0.002284:	‡	1401100.0- :E0
304: -0.002251:	‡	1403100.0- :E0

44-230075

270365-45

Non. Lin. Error, SN = 4130 (Continued)

305: -0.002417:	*	1009100.0-100
306: -0.002269:	*	1009000.0-100
307: -0.002483:	*	1008900.0-100
308: -0.002265:	*	1008800.0-100
309: -0.002589:	*	1008700.0-100
30A: -0.002383:	*	1008600.0-100
30B: -0.002508:	*	1008500.0-100
30C: -0.002336:	*	1008400.0-100
30D: -0.002560:	*	1008300.0-100
30E: -0.002428:	*	1008200.0-100
30F: -0.002677:	*	1008100.0-100
310: -0.002528:	*	1008000.0-100
311: -0.002861:	*	1007900.0-100
312: -0.002612:	*	1007800.0-100
313: -0.002746:	*	1007700.0-100
314: -0.002710:	*	1007600.0-100
315: -0.002955:	*	1007500.0-100
316: -0.002813:	*	1007400.0-100
317: -0.002864:	*	1007300.0-100
318: -0.002770:	*	1007200.0-100
319: -0.002959:	*	1007100.0-100
31A: -0.002888:	*	1007000.0-100
31B: -0.002901:	*	1006900.0-100
31C: -0.002742:	*	1006800.0-100
31D: -0.002975:	*	1006700.0-100
31E: -0.002878:	*	1006600.0-100
31F: -0.003165:	*	1006500.0-100
320: -0.002991:	*	1006400.0-100
321: -0.003220:	*	1006300.0-100
322: -0.003083:	*	1006200.0-100
323: -0.003195:	*	1006100.0-100
324: -0.003109:	*	1006000.0-100
325: -0.003314:	*	1005900.0-100
326: -0.003130:	*	1005800.0-100
327: -0.003246:	*	1005700.0-100
328: -0.003301:	*	1005600.0-100
329: -0.003397:	*	1005500.0-100
32A: -0.003247:	*	1005400.0-100
32B: -0.003362:	*	1005300.0-100
32C: -0.002182:	*	1005200.0-100
32D: -0.002338:	*	1005100.0-100
32E: -0.002251:	*	1005000.0-100
32F: -0.002332:	*	1004900.0-100
330: -0.001979:	*	1004800.0-100
331: -0.002225:	*	1004700.0-100
332: -0.002099:	*	1004600.0-100
333: -0.002164:	*	1004500.0-100
334: -0.001894:	*	1004400.0-100
335: -0.002134:	*	1004300.0-100
336: -0.002018:	*	1004200.0-100
337: -0.002019:	*	1004100.0-100
338: -0.001991:	*	1004000.0-100
339: -0.002182:	*	1003900.0-100
33A: -0.002183:	*	1003800.0-100
33B: -0.002134:	*	1003700.0-100
33C: -0.002005:	*	1003600.0-100
33D: -0.002338:	*	1003500.0-100
33E: -0.002115:	*	1003400.0-100
33F: -0.002380:	*	1003300.0-100
340: -0.000653:	*	1003200.0-100

TA-000005

270365-46

Non. Lin. Error, SN = 4130 (Continued)

341: -0.001006:	* :P0000.0-10
342: -0.000888:	* :P0000.0-10
343: -0.001175:	* :P0000.0-10
344: -0.000834:	* :P0000.0-10
345: -0.000951:	* :P0000.0-10
346: -0.001030:	* :P0000.0-10
347: -0.000951:	* :P0000.0-10
348: -0.000710:	* :P0000.0-10
349: -0.000672:	* :P0000.0-10
34A: -0.000854:	* :P0000.0-10
34B: -0.000934:	* :P0000.0-10
34C: -0.000648:	* :P0000.0-10
34D: -0.001008:	* :P0000.0-10
34E: -0.000828:	* :P0000.0-10
34F: -0.000777:	* :P0000.0-10
350: -0.000774:	* :P0000.0-10
351: -0.000994:	* :P0000.0-10
352: -0.000901:	* :P0000.0-10
353: -0.000985:	* :P0000.0-10
354: -0.000618:	* :P0000.0-10
355: -0.000920:	* :P0000.0-10
356: -0.000731:	* :P0000.0-10
357: -0.000707:	* :P0000.0-10
358: -0.000832:	* :P0000.0-10
359: -0.001047:	* :P0000.0-10
35A: -0.001003:	* :P0000.0-10
35B: -0.000976:	* :P0000.0-10
35C: -0.000957:	* :P0000.0-10
35D: -0.001273:	* :P0000.0-10
35E: -0.000994:	* :P0000.0-10
35F: -0.001038:	* :P0000.0-10
360: -0.001150:	* :P0000.0-10
361: -0.001320:	* :P0000.0-10
362: -0.001257:	* :P0000.0-10
363: -0.001390:	* :P0000.0-10
364: -0.001263:	* :P0000.0-10
365: -0.001498:	* :P0000.0-10
366: -0.001388:	* :P0000.0-10
367: -0.001453:	* :P0000.0-10
368: -0.001295:	* :P0000.0-10
369: -0.001416:	* :P0000.0-10
36A: -0.001389:	* :P0000.0-10
36B: -0.001498:	* :P0000.0-10
36C: -0.001502:	* :P0000.0-10
36D: -0.001797:	* :P0000.0-10
36E: -0.001566:	* :P0000.0-10
36F: -0.001596:	* :P0000.0-10
370: -0.001531:	* :P0000.0-10
371: -0.001799:	* :P0000.0-10
372: -0.001684:	* :P0000.0-10
373: -0.001735:	* :P0000.0-10
374: -0.001647:	* :P0000.0-10
375: -0.001857:	* :P0000.0-10
376: -0.001809:	* :P0000.0-10
377: -0.001697:	* :P0000.0-10
378: -0.001770:	* :P0000.0-10
379: -0.001956:	* :P0000.0-10
37A: -0.001821:	* :P0000.0-10
37B: -0.001841:	* :P0000.0-10
37C: -0.001820:	* :P0000.0-10

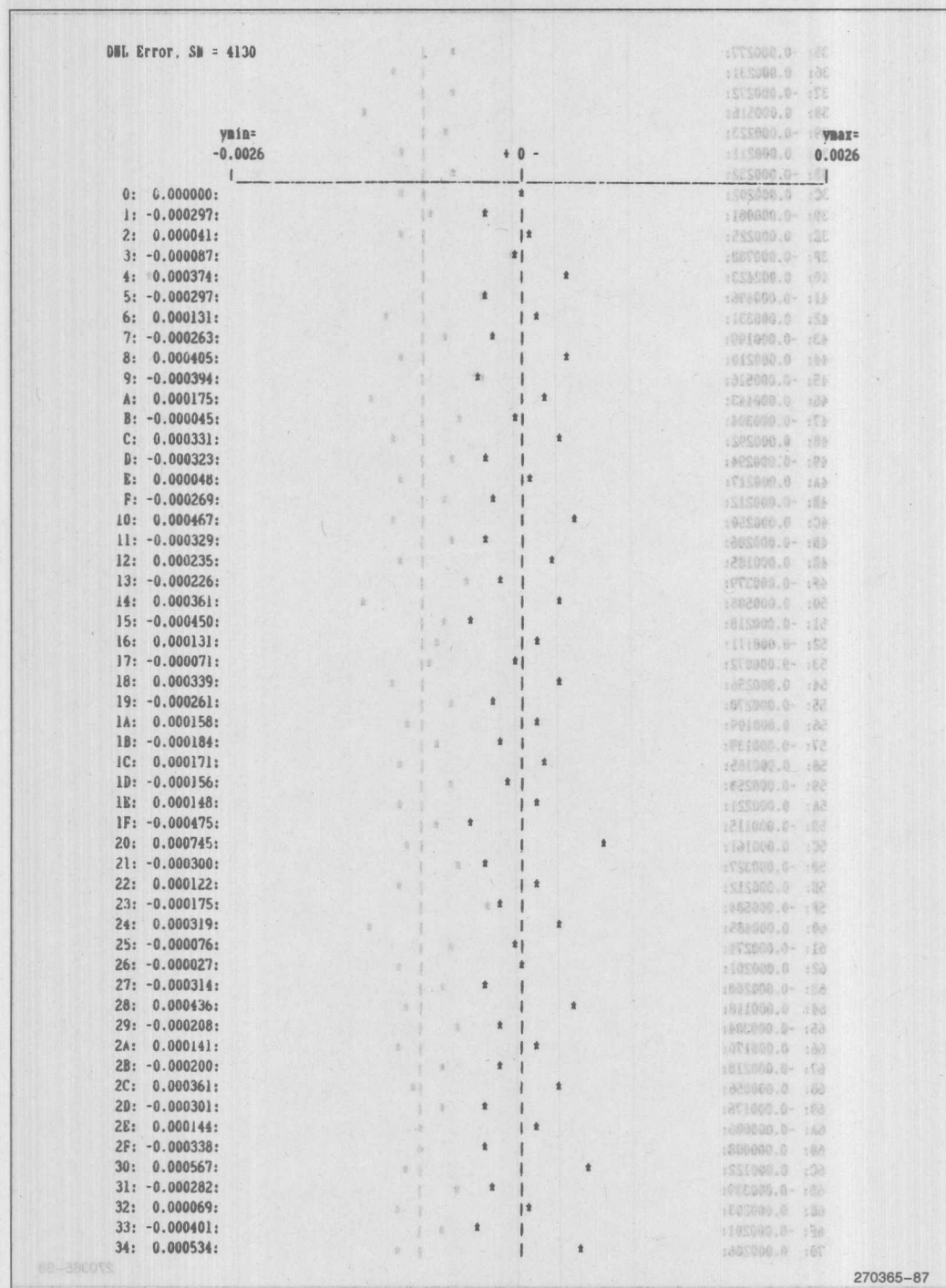
6-268013

270365-47

Non. Lin. Error, SN = 4130 (Continued)

37D: -0.001979:	*	17800.0 -190	
37E: -0.002106:	*	18500.0 -180	
37F: -0.001597:	*	19200.0 -170	
380: -0.001784:	*	19900.0 -160	
381: -0.002005:	*	20600.0 -150	
382: -0.001840:	*	21300.0 -140	
383: -0.001907:	*	22000.0 -130	
384: -0.001890:	*	22700.0 -120	
385: -0.001989:	*	23400.0 -110	
386: -0.001867:	*	24100.0 -100	
387: -0.001957:	*	24800.0 -90	
388: -0.002029:	*	25500.0 -80	
389: -0.002256:	*	26200.0 -70	
38A: -0.002177:	*	26900.0 -60	
38B: -0.002205:	*	27600.0 -50	
38C: -0.002294:	*	28300.0 -40	
38D: -0.002497:	*	29000.0 -30	
38E: -0.002204:	*	29700.0 -20	
38F: -0.002499:	*	30400.0 -10	
390: -0.002201:	*	31100.0 0	
391: -0.002774:	*	31800.0 10	
392: -0.002398:	*	32500.0 20	
393: -0.002591:	*	33200.0 30	
394: -0.002325:	*	33900.0 40	
395: -0.002570:	*	34600.0 50	
396: -0.002590:	*	35300.0 60	
397: -0.002513:	*	36000.0 70	
398: -0.002409:	*	36700.0 80	
399: -0.002662:	*	37400.0 90	
39A: -0.002513:	*	38100.0 100	
39B: -0.002588:	*	38800.0 110	
39C: -0.002345:	*	39500.0 120	
39D: -0.002633:	*	40200.0 130	
39E: -0.002485:	*	40900.0 140	
39F: -0.002579:	*	41600.0 150	
3A0: -0.002427:	*	42300.0 160	
3A1: -0.002646:	*	43000.0 170	
3A2: -0.002572:	*	43700.0 180	
3A3: -0.002639:	*	44400.0 190	
3A4: -0.002393:	*	45100.0 200	
3A5: -0.002494:	*	45800.0 210	
3A6: -0.002609:	*	46500.0 220	
3A7: -0.002570:	*	47200.0 230	
3A8: -0.002522:	*	47900.0 240	
3A9: -0.002809:	*	48600.0 250	
3AA: -0.002658:	*	49300.0 260	
3AB: -0.002698:	*	50000.0 270	
3AC: -0.002645:	*	50700.0 280	
3AD: -0.002724:	*	51400.0 290	
3AE: -0.002721:	*	52100.0 300	
3AF: -0.002685:	*	52800.0 310	
3B0: -0.002752:	*	53500.0 320	
3B1: -0.003015:	*	54200.0 330	
3B2: -0.002837:	*	54900.0 340	
3B3: -0.002932:	*	55600.0 350	
3B4: -0.002689:	*	56300.0 360	
3B5: -0.003034:	*	57000.0 370	
3B6: -0.002817:	*	57700.0 380	
3B7: -0.002812:	*	58400.0 390	
3B8: -0.002965:	*	59100.0 400	

270365-50



35: -0.000277:
 36: 0.000231:
 37: -0.000272:
 38: 0.000516:
 39: -0.000223:
 3A: 0.000211:
 3B: -0.000232:
 3C: 0.000202:
 3D: -0.000081:
 3E: 0.000225:
 3F: -0.000788:
 40: 0.002423:
 41: -0.000496:
 42: 0.000331:
 43: -0.000199:
 44: 0.000210:
 45: -0.000516:
 46: 0.000443:
 47: -0.000304:
 48: 0.000292:
 49: -0.000294:
 4A: 0.000217:
 4B: -0.000212:
 4C: 0.000250:
 4D: -0.000286:
 4E: 0.000185:
 4F: -0.000379:
 50: 0.000508:
 51: -0.000218:
 52: -0.000111:
 53: -0.000072:
 54: 0.000256:
 55: -0.000270:
 56: 0.000109:
 57: -0.000139:
 58: 0.000185:
 59: -0.000258:
 5A: 0.000221:
 5B: -0.000115:
 5C: 0.000161:
 5D: -0.000327:
 5E: 0.000212:
 5F: -0.000584:
 60: 0.000485:
 61: -0.000274:
 62: 0.000201:
 63: -0.000208:
 64: 0.000118:
 65: -0.000304:
 66: 0.000170:
 67: -0.000218:
 68: 0.000056:
 69: -0.000176:
 6A: -0.000006:
 6B: 0.000008:
 6C: 0.000122:
 6D: -0.000339:
 6E: 0.000203:
 6F: -0.000201:
 70: 0.000206:



0C19 = 42, 10000 380

-0.004
0000.0

000000.0- 10
 000000.0- 11
 000000.0- 12
 000000.0- 13
 000000.0- 14
 000000.0- 15
 000000.0- 16
 000000.0- 17
 000000.0- 18
 000000.0- 19
 000000.0- 20
 000000.0- 21
 000000.0- 22
 000000.0- 23
 000000.0- 24
 000000.0- 25
 000000.0- 26
 000000.0- 27
 000000.0- 28
 000000.0- 29
 000000.0- 30
 000000.0- 31
 000000.0- 32
 000000.0- 33
 000000.0- 34
 000000.0- 35
 000000.0- 36
 000000.0- 37
 000000.0- 38
 000000.0- 39
 000000.0- 40
 000000.0- 41
 000000.0- 42
 000000.0- 43
 000000.0- 44
 000000.0- 45
 000000.0- 46
 000000.0- 47
 000000.0- 48
 000000.0- 49
 000000.0- 50
 000000.0- 51
 000000.0- 52
 000000.0- 53
 000000.0- 54
 000000.0- 55
 000000.0- 56
 000000.0- 57
 000000.0- 58
 000000.0- 59
 000000.0- 60
 000000.0- 61
 000000.0- 62
 000000.0- 63
 000000.0- 64
 000000.0- 65
 000000.0- 66
 000000.0- 67
 000000.0- 68
 000000.0- 69
 000000.0- 70

TS-200013

270365-88

DNL Error, SN = 4130 (Continued)

[illegible]

270365-89

6-273

6

AD: -0.000273:	*		*	104000.0 -117
AE: 0.000137:	*		*	105000.0 -117
AF: -0.000194:	*		*	106000.0 -117
B0: 0.000366:	*		*	107000.0 -117
B1: -0.000233:	*		*	108000.0 -117
B2: 0.000111:	*		*	109000.0 -117
B3: -0.000115:	*		*	110000.0 -117
B4: -0.000039:	*		*	111000.0 -117
B5: -0.000088:	*		*	112000.0 -117
B6: 0.000100:	*		*	113000.0 -117
B7: -0.000147:	*		*	114000.0 -117
B8: 0.000109:	*		*	115000.0 -117
B9: -0.000171:	*		*	116000.0 -117
BA: 0.000156:	*		*	117000.0 -117
BB: -0.000180:	*		*	118000.0 -117
BC: 0.000041:	*		*	119000.0 -117
BD: -0.000134:	*		*	120000.0 -117
BE: 0.000202:	*		*	121000.0 -117
BF: -0.000561:	*		*	122000.0 -117
C0: 0.002134:	*		*	123000.0 -117
C1: -0.000301:	*		*	124000.0 -117
C2: 0.000150:	*		*	125000.0 -117
C3: -0.000138:	*		*	126000.0 -117
C4: 0.000206:	*		*	127000.0 -117
C5: -0.000371:	*		*	128000.0 -117
C6: 0.000377:	*		*	129000.0 -117
C7: -0.000341:	*		*	130000.0 -117
C8: 0.000272:	*		*	131000.0 -117
C9: -0.000235:	*		*	132000.0 -117
CA: -0.000105:	*		*	133000.0 -117
CB: 0.000091:	*		*	134000.0 -117
CC: 0.000203:	*		*	135000.0 -117
CD: -0.000322:	*		*	136000.0 -117
CE: 0.000207:	*		*	137000.0 -117
CF: -0.000187:	*		*	138000.0 -117
D0: 0.000151:	*		*	139000.0 -117
D1: -0.000250:	*		*	140000.0 -117
D2: 0.000142:	*		*	141000.0 -117
D3: -0.000396:	*		*	142000.0 -117
D4: 0.000501:	*		*	143000.0 -117
D5: -0.000362:	*		*	144000.0 -117
D6: 0.000329:	*		*	145000.0 -117
D7: -0.000169:	*		*	146000.0 -117
D8: 0.000029:	*		*	147000.0 -117
D9: -0.000115:	*		*	148000.0 -117
DA: 0.000186:	*		*	149000.0 -117
DB: -0.000113:	*		*	150000.0 -117
DC: -0.000010:	*		*	151000.0 -117
DD: -0.000287:	*		*	152000.0 -117
DE: 0.000153:	*		*	153000.0 -117
DF: -0.000331:	*		*	154000.0 -117
E0: 0.000308:	*		*	155000.0 -117
E1: -0.000506:	*		*	156000.0 -117
E2: 0.000409:	*		*	157000.0 -117
E3: -0.000184:	*		*	158000.0 -117
E4: -0.000124:	*		*	159000.0 -117
E5: 0.000020:	*		*	160000.0 -117
E6: 0.000181:	*		*	161000.0 -117
E7: -0.000145:	*		*	162000.0 -117
E8: 0.000210:	*		*	163000.0 -117

88-280073

270365-90

DNL Error, SN = 4130 (Continued)

E9: -0.000232:	+		:00000.0- :001
EA: 0.000136:			:00100.0- :001
EB: -0.000151:	+		:00200.0- :001
EC: 0.000168:			:00300.0- :001
ED: -0.000212:	+		:00400.0- :001
EE: 0.000195:			:00500.0- :001
EF: -0.000171:	+		:00600.0- :001
FO: 0.000115:			:00700.0- :001
F1: -0.000376:	+		:00800.0- :001
F2: 0.000208:			:00900.0- :001
F3: -0.000167:	+		:01000.0- :001
F4: 0.000078:			:01100.0- :001
F5: -0.000287:	+		:01200.0- :001
F6: 0.000348:			:01300.0- :001
F7: -0.000231:	+		:01400.0- :001
F8: -0.000161:	+		:01500.0- :001
F9: -0.000022:			:01600.0- :001
FA: 0.000265:			:01700.0- :001
FB: -0.000130:	+		:01800.0- :001
FC: 0.000167:			:01900.0- :001
FD: -0.000144:	+		:02000.0- :001
FE: 0.000086:			:02100.0- :001
FF: -0.000880:			:02200.0- :001
100: 0.001348:			:02300.0- :001
101: -0.000334:	+		:02400.0- :001
102: 0.000236:			:02500.0- :001
103: -0.000222:	+		:02600.0- :001
104: -0.000030:			:02700.0- :001
105: -0.000123:	+		:02800.0- :001
106: 0.000008:			:02900.0- :001
107: -0.000068:	+		:03000.0- :001
108: 0.000266:			:03100.0- :001
109: -0.000265:	+		:03200.0- :001
10A: 0.000136:			:03300.0- :001
10B: -0.000078:	+		:03400.0- :001
10C: 0.000069:			:03500.0- :001
10D: -0.000260:	+		:03600.0- :001
10E: 0.000142:			:03700.0- :001
10F: -0.000352:	+		:03800.0- :001
110: 0.000142:			:03900.0- :001
111: -0.000146:	+		:04000.0- :001
112: 0.000292:			:04100.0- :001
113: -0.000150:	+		:04200.0- :001
114: 0.000003:			:04300.0- :001
115: -0.000263:	+		:04400.0- :001
116: 0.000174:			:04500.0- :001
117: -0.000151:	+		:04600.0- :001
118: 0.000078:			:04700.0- :001
119: -0.000120:	+		:04800.0- :001
11A: 0.000027:			:04900.0- :001
11B: -0.000120:	+		:05000.0- :001
11C: 0.000048:			:05100.0- :001
11D: -0.000248:	+		:05200.0- :001
11E: 0.000192:			:05300.0- :001
11F: -0.000455:	+		:05400.0- :001
120: 0.000535:			:05500.0- :001
121: -0.000300:	+		:05600.0- :001
122: 0.000139:			:05700.0- :001
123: -0.000088:	+		:05800.0- :001
124: 0.000145:			:05900.0- :001

20-20005

270365-91

DNL Error, SN = 4130 (Continued)

126: 0.000193:
 127: -0.000028:
 128: 0.000202:
 129: -0.000194:
 12A: 0.000192:
 12B: -0.000114:
 12C: -0.000060:
 12D: -0.000148:
 12E: 0.000048:
 12F: -0.000100:
 130: 0.000168:
 131: -0.000263:
 132: 0.000188:
 133: -0.000178:
 134: 0.000193:
 135: -0.000303:
 136: 0.000259:
 137: -0.000250:
 138: 0.000436:
 139: -0.000228:
 13A: -0.000052:
 13B: 0.000008:
 13C: 0.000271:
 13D: -0.000245:
 13E: 0.000123:
 13F: -0.000471:
 140: 0.001823:
 141: -0.000033:
 142: 0.000139:
 143: -0.000132:
 144: 0.000199:
 145: -0.000231:
 146: 0.000116:
 147: -0.000051:
 148: 0.000217:
 149: -0.000271:
 14A: 0.000134:
 14B: 0.000060:
 14C: -0.000010:
 14D: -0.000225:
 14E: 0.000102:
 14F: -0.000207:
 150: 0.000168:
 151: -0.000191:
 152: 0.000133:
 153: -0.000171:
 154: 0.000170:
 155: -0.000239:
 156: 0.000056:
 157: -0.000001:
 158: 0.000199:
 159: -0.000277:
 15A: 0.000110:
 15B: -0.000074:
 15C: 0.000104:
 15D: -0.000207:
 15E: 0.000160:
 15F: -0.000226:
 160: 0.000138:

126: 0.000193:
 127: -0.000028:
 128: 0.000202:
 129: -0.000194:
 12A: 0.000192:
 12B: -0.000114:
 12C: -0.000060:
 12D: -0.000148:
 12E: 0.000048:
 12F: -0.000100:
 130: 0.000168:
 131: -0.000263:
 132: 0.000188:
 133: -0.000178:
 134: 0.000193:
 135: -0.000303:
 136: 0.000259:
 137: -0.000250:
 138: 0.000436:
 139: -0.000228:
 13A: -0.000052:
 13B: 0.000008:
 13C: 0.000271:
 13D: -0.000245:
 13E: 0.000123:
 13F: -0.000471:
 140: 0.001823:
 141: -0.000033:
 142: 0.000139:
 143: -0.000132:
 144: 0.000199:
 145: -0.000231:
 146: 0.000116:
 147: -0.000051:
 148: 0.000217:
 149: -0.000271:
 14A: 0.000134:
 14B: 0.000060:
 14C: -0.000010:
 14D: -0.000225:
 14E: 0.000102:
 14F: -0.000207:
 150: 0.000168:
 151: -0.000191:
 152: 0.000133:
 153: -0.000171:
 154: 0.000170:
 155: -0.000239:
 156: 0.000056:
 157: -0.000001:
 158: 0.000199:
 159: -0.000277:
 15A: 0.000110:
 15B: -0.000074:
 15C: 0.000104:
 15D: -0.000207:
 15E: 0.000160:
 15F: -0.000226:
 160: 0.000138:

126: 0.000193:
 127: -0.000028:
 128: 0.000202:
 129: -0.000194:
 12A: 0.000192:
 12B: -0.000114:
 12C: -0.000060:
 12D: -0.000148:
 12E: 0.000048:
 12F: -0.000100:
 130: 0.000168:
 131: -0.000263:
 132: 0.000188:
 133: -0.000178:
 134: 0.000193:
 135: -0.000303:
 136: 0.000259:
 137: -0.000250:
 138: 0.000436:
 139: -0.000228:
 13A: -0.000052:
 13B: 0.000008:
 13C: 0.000271:
 13D: -0.000245:
 13E: 0.000123:
 13F: -0.000471:
 140: 0.001823:
 141: -0.000033:
 142: 0.000139:
 143: -0.000132:
 144: 0.000199:
 145: -0.000231:
 146: 0.000116:
 147: -0.000051:
 148: 0.000217:
 149: -0.000271:
 14A: 0.000134:
 14B: 0.000060:
 14C: -0.000010:
 14D: -0.000225:
 14E: 0.000102:
 14F: -0.000207:
 150: 0.000168:
 151: -0.000191:
 152: 0.000133:
 153: -0.000171:
 154: 0.000170:
 155: -0.000239:
 156: 0.000056:
 157: -0.000001:
 158: 0.000199:
 159: -0.000277:
 15A: 0.000110:
 15B: -0.000074:
 15C: 0.000104:
 15D: -0.000207:
 15E: 0.000160:
 15F: -0.000226:
 160: 0.000138:

270365-92

DNL Error, SN = 4130 (Continued)

270365-93

19D: -0.000329:	* :175000.0- :181
19E: -0.000018:	* :175000.0- :181
19F: -0.000045:	* :175000.0- :181
1A0: 0.000210:	* :175000.0- :181
1A1: -0.000244:	* :175000.0- :181
1A2: 0.000139:	* :175000.0- :181
1A3: -0.000091:	* :175000.0- :181
1A4: -0.000041:	* :175000.0- :181
1A5: -0.000210:	* :175000.0- :181
1A6: 0.000162:	* :175000.0- :181
1A7: -0.000057:	* :175000.0- :181
1A8: 0.000163:	* :175000.0- :181
1A9: -0.000158:	* :175000.0- :181
1AA: 0.000114:	* :175000.0- :181
1AB: 0.000024:	* :175000.0- :181
1AC: 0.000028:	* :175000.0- :181
1AD: -0.000240:	* :175000.0- :181
1AE: 0.000110:	* :175000.0- :181
1AF: 0.000189:	* :175000.0- :181
1B0: -0.000014:	* :175000.0- :181
1B1: -0.000229:	* :175000.0- :181
1B2: 0.000134:	* :175000.0- :181
1B3: -0.000075:	* :175000.0- :181
1B4: 0.000018:	* :175000.0- :181
1B5: -0.000135:	* :175000.0- :181
1B6: 0.000041:	* :175000.0- :181
1B7: 0.000053:	* :175000.0- :181
1B8: -0.000040:	* :175000.0- :181
1B9: -0.000060:	* :175000.0- :181
1BA: 0.000200:	* :175000.0- :181
1BB: -0.000037:	* :175000.0- :181
1BC: -0.000024:	* :175000.0- :181
1BD: -0.000169:	* :175000.0- :181
1BE: 0.000088:	* :175000.0- :181
1BF: 0.000013:	* :175000.0- :181
1C0: 0.001412:	* :175000.0- :181
1C1: -0.000118:	* :175000.0- :181
1C2: 0.000178:	* :175000.0- :181
1C3: -0.000132:	* :175000.0- :181
1C4: 0.000203:	* :175000.0- :181
1C5: -0.000248:	* :175000.0- :181
1C6: 0.000212:	* :175000.0- :181
1C7: -0.000144:	* :175000.0- :181
1C8: 0.000258:	* :175000.0- :181
1C9: -0.000309:	* :175000.0- :181
1CA: 0.000028:	* :175000.0- :181
1CB: 0.000056:	* :175000.0- :181
1CC: 0.000133:	* :175000.0- :181
1CD: -0.000250:	* :175000.0- :181
1CE: 0.000083:	* :175000.0- :181
1CF: -0.000002:	* :175000.0- :181
1D0: 0.000169:	* :175000.0- :181
1D1: -0.000269:	* :175000.0- :181
1D2: 0.000102:	* :175000.0- :181
1D3: -0.000051:	* :175000.0- :181
1D4: 0.000168:	* :175000.0- :181
1D5: -0.000210:	* :175000.0- :181
1D6: 0.000007:	* :175000.0- :181
1D7: 0.000115:	* :175000.0- :181
1D8: 0.000064:	* :175000.0- :181

DP-38471

270365-94

DNL Error, SN = 4130 (Continued)

1D9: -0.000256:	*		1D9000.0	-215
1DA: 0.000086:		*	1D1000.0	-165
1DB: -0.000008:	*		1D0000.0	-115
1DC: 0.000042:		*	1C9000.0	-65
1DD: -0.000101:	*		1C8000.0	-15
1DE: 0.000115:		*	1C7000.0	-35
1DF: -0.000013:	*		1C6000.0	-15
1E0: 0.000050:		*	1C5000.0	-15
1E1: -0.000396:	*		1C4000.0	-65
1E2: 0.000231:		*	1C3000.0	-115
1E3: -0.000279:	*		1C2000.0	-115
1E4: 0.000305:		*	1C1000.0	-65
1E5: -0.000235:	*		1C0000.0	-15
1E6: 0.000233:		*	1B9000.0	-155
1E7: -0.000039:	*		1B8000.0	-155
1E8: 0.000140:		*	1B7000.0	-155
1E9: -0.000454:	*		1B6000.0	-205
1EA: 0.000295:		*	1B5000.0	-105
1EB: -0.000104:	*		1B4000.0	-155
1EC: 0.000031:		*	1B3000.0	-155
1ED: -0.000208:	*		1B2000.0	-155
1EE: -0.000063:		*	1B1000.0	-105
1EF: 0.000209:		*	1B0000.0	-155
1F0: 0.000041:		*	1A9000.0	-105
1F1: -0.000225:	*		1A8000.0	-105
1F2: 0.000094:		*	1A7000.0	-105
1F3: 0.000023:	*		1A6000.0	-105
1F4: 0.000025:		*	1A5000.0	-105
1F5: -0.000064:	*		1A4000.0	-115
1F6: -0.000011:		*	1A3000.0	-115
1F7: 0.000150:		*	1A2000.0	-115
1F8: -0.000031:	*		1A1000.0	-105
1F9: -0.000186:	*		1A0000.0	-105
1FA: 0.000116:		*	199000.0	-105
1FB: -0.000038:	*		198000.0	-115
1FC: 0.000073:		*	197000.0	-115
1FD: -0.000222:	*		196000.0	-115
1FE: 0.000090:		*	195000.0	-115
1FF: -0.000671:	*		194000.0	-115
200: 0.000468:		*	193000.0	-115
201: -0.000223:	*		192000.0	-115
202: 0.000196:		*	191000.0	-115
203: -0.000249:	*		190000.0	-115
204: 0.000099:		*	189000.0	-105
205: -0.000048:	*		188000.0	-115
206: 0.000070:		*	187000.0	-115
207: -0.000189:	*		186000.0	-115
208: -0.000011:		*	185000.0	-115
209: -0.000307:	*		184000.0	-115
20A: 0.000429:		*	183000.0	-115
20B: -0.000207:	*		182000.0	-115
20C: -0.000085:		*	181000.0	-105
20D: -0.000109:	*		180000.0	-105
20E: -0.000034:		*	179000.0	-115
20F: -0.000197:	*		178000.0	-115
210: 0.000420:		*	177000.0	-115
211: -0.000332:	*		176000.0	-115
212: 0.000142:		*	175000.0	-115
213: -0.000076:	*		174000.0	-115
214: 0.000275:		*	173000.0	-115

215: -0.000309:	*		:25000.0-	:001
216: 0.000159:			:20000.0-	:001
217: -0.000056:	*		:00000.0-	:001
218: 0.000026:	*		:50000.0-	:001
219: -0.000200:	*		:01000.0-	:001
21A: -0.000012:	*		:21000.0-	:001
21B: 0.000082:	*		:01000.0-	:001
21C: 0.000379:			:00100.0-	:001
21D: -0.000487:	*		:00000.0-	:001
21E: 0.000199:			:40000.0-	:001
21F: -0.000551:	*		:00000.0-	:001
220: 0.000536:			:00100.0-	:001
221: -0.000267:	*		:20000.0-	:001
222: 0.000167:			:40000.0-	:001
223: -0.000142:	*		:00000.0-	:001
224: 0.000111:	*		:00000.0-	:001
225: -0.000151:	*		:00000.0-	:001
226: 0.000040:			:00000.0-	:001
227: -0.000104:	*		:00000.0-	:001
228: 0.000333:			:10000.0-	:001
229: -0.000305:	*		:00000.0-	:001
22A: 0.000125:	*		:00000.0-	:001
22B: -0.000026:	*		:00000.0-	:001
22C: 0.000065:	*		:00000.0-	:001
22D: -0.000182:	*		:00000.0-	:001
22E: 0.000260:			:00000.0-	:001
22F: -0.000362:	*		:00000.0-	:001
230: 0.000374:			:00000.0-	:001
231: -0.000245:	*		:00000.0-	:001
232: 0.000281:			:00000.0-	:001
233: -0.000276:	*		:00000.0-	:001
234: 0.000223:			:00000.0-	:001
235: -0.000173:	*		:00000.0-	:001
236: -0.000079:	*		:00000.0-	:001
237: 0.000098:	*		:00000.0-	:001
238: 0.000180:			:00000.0-	:001
239: -0.000481:	*		:00000.0-	:001
23A: 0.000359:			:00000.0-	:001
23B: -0.000143:	*		:00000.0-	:001
23C: 0.000161:			:00000.0-	:001
23D: -0.000188:	*		:00000.0-	:001
23E: 0.000104:	*		:00000.0-	:001
23F: -0.000568:	*		:00000.0-	:001
240: 0.001646:			:00000.0-	:001
241: -0.000170:	*		:00000.0-	:001
242: 0.000226:			:00000.0-	:001
243: -0.000170:	*		:00000.0-	:001
244: 0.000133:	*		:00000.0-	:001
245: -0.000295:	*		:00000.0-	:001
246: 0.000305:			:00000.0-	:001
247: -0.000255:	*		:00000.0-	:001
248: 0.000187:			:00000.0-	:001
249: -0.000053:	*		:00000.0-	:001
24A: 0.000090:	*		:00000.0-	:001
24B: -0.000091:	*		:00000.0-	:001
24C: 0.000255:			:00000.0-	:001
24D: -0.000233:	*		:00000.0-	:001
24E: 0.000108:	*		:00000.0-	:001
24F: -0.000056:	*		:00000.0-	:001
250: 0.000273:	*		:00000.0-	:001

50-20000

270365-96

DNL Error, SN = 4130 (Continued)

270365-97

```

28D: -0.000032:  * 000000.0- 1000
28E: 0.000077:  | 000000.0- 1000
28F: -0.000164:  * 000000.0- 1000
290: 0.000070:  | 000000.0- 1000
291: -0.000136:  * 000000.0- 1000
292: 0.000062:  | 000000.0- 1000
293: -0.000183:  * 000000.0- 1000
294: 0.000166:  | 000000.0- 1000
295: -0.000273:  * 000000.0- 1000
296: 0.000133:  | 000000.0- 1000
297: -0.000076:  * 000000.0- 1000
298: 0.000045:  | 000000.0- 1000
299: -0.000234:  * 000000.0- 1000
29A: -0.000066:  * 000000.0- 1000
29B: 0.000072:  | 000000.0- 1000
29C: 0.000123:  | 000000.0- 1000
29D: -0.000304:  * 000000.0- 1000
29E: 0.000137:  | 000000.0- 1000
29F: -0.000206:  * 000000.0- 1000
2A0: 0.000196:  | 000000.0- 1000
2A1: -0.000150:  * 000000.0- 1000
2A2: 0.000181:  | 000000.0- 1000
2A3: -0.000160:  * 000000.0- 1000
2A4: 0.000106:  | 000000.0- 1000
2A5: -0.000262:  * 000000.0- 1000
2A6: 0.000167:  | 000000.0- 1000
2A7: -0.000123:  * 000000.0- 1000
2A8: 0.000193:  | 000000.0- 1000
2A9: -0.000283:  * 000000.0- 1000
2AA: -0.000117:  * 000000.0- 1000
2AB: 0.000193:  | 000000.0- 1000
2AC: 0.000140:  | 000000.0- 1000
2AD: -0.000215:  * 000000.0- 1000
2AE: -0.000044:  * 000000.0- 1000
2AF: 0.000047:  | 000000.0- 1000
2B0: 0.000028:  * 000000.0- 1000
2B1: -0.000322:  * 000000.0- 1000
2B2: 0.000207:  | 000000.0- 1000
2B3: -0.000082:  * 000000.0- 1000
2B4: 0.000125:  | 000000.0- 1000
2B5: -0.000239:  * 000000.0- 1000
2B6: 0.000078:  | 000000.0- 1000
2B7: 0.000017:  * 000000.0- 1000
2B8: 0.000128:  | 000000.0- 1000
2B9: -0.000179:  * 000000.0- 1000
2BA: 0.000101:  | 000000.0- 1000
2BB: -0.000071:  * 000000.0- 1000
2BC: 0.000359:  | 000000.0- 1000
2BD: -0.000248:  * 000000.0- 1000
2BE: 0.000054:  | 000000.0- 1000
2BF: 0.000079:  | 000000.0- 1000
2C0: 0.001402:  | 000000.0- 1000
2C1: -0.000261:  * 000000.0- 1000
2C2: 0.000127:  | 000000.0- 1000
2C3: -0.000113:  * 000000.0- 1000
2C4: 0.000268:  | 000000.0- 1000
2C5: -0.000143:  * 000000.0- 1000
2C6: 0.000007:  * 000000.0- 1000
2C7: 0.000027:  * 000000.0- 1000
2C8: 0.000243:  | 000000.0- 1000

```

TG-326015

270365-98

DNL Error, SN = 4130 (Continued)

2C9: -0.000312:	±		±01000.0-1000
2CA: 0.000040:	±		±01000.0-1000
2CB: 0.000043:	±		±01000.0-1000
2CC: 0.000164:	±		±01000.0-1000
2CD: -0.000183:	±		±01000.0-1000
2CE: -0.000021:	±		±01000.0-1000
2CF: -0.000126:	±		±01000.0-1000
2D0: 0.000365:	±		±01000.0-1000
2D1: -0.000336:	±		±01000.0-1000
2D2: 0.000091:	±		±01000.0-1000
2D3: -0.000143:	±		±01000.0-1000
2D4: 0.000220:	±		±01000.0-1000
2D5: -0.000288:	±		±01000.0-1000
2D6: 0.000251:	±		±01000.0-1000
2D7: 0.000001:	±		±01000.0-1000
2D8: -0.000069:	±		±01000.0-1000
2D9: -0.000222:	±		±01000.0-1000
2DA: 0.000336:	±		±01000.0-1000
2DB: -0.000071:	±		±01000.0-1000
2DC: 0.000122:	±		±01000.0-1000
2DD: -0.000540:	±		±01000.0-1000
2DE: 0.000384:	±		±01000.0-1000
2DF: -0.000122:	±		±01000.0-1000
2E0: -0.000134:	±		±01000.0-1000
2E1: -0.000114:	±		±01000.0-1000
2E2: 0.000073:	±		±01000.0-1000
2E3: -0.000040:	±		±01000.0-1000
2E4: -0.000188:	±		±01000.0-1000
2E5: 0.000033:	±		±01000.0-1000
2E6: 0.000173:	±		±01000.0-1000
2E7: -0.000049:	±		±01000.0-1000
2E8: 0.000019:	±		±01000.0-1000
2E9: -0.000234:	±		±01000.0-1000
2EA: 0.000176:	±		±01000.0-1000
2EB: -0.000281:	±		±01000.0-1000
2EC: 0.000200:	±		±01000.0-1000
2ED: -0.000174:	±		±01000.0-1000
2EE: 0.000042:	±		±01000.0-1000
2EF: -0.000086:	±		±01000.0-1000
2F0: 0.000063:	±		±01000.0-1000
2F1: -0.000160:	±		±01000.0-1000
2F2: 0.000127:	±		±01000.0-1000
2F3: -0.000134:	±		±01000.0-1000
2F4: 0.000058:	±		±01000.0-1000
2F5: -0.000211:	±		±01000.0-1000
2F6: 0.000104:	±		±01000.0-1000
2F7: -0.000031:	±		±01000.0-1000
2F8: -0.000134:	±		±01000.0-1000
2F9: -0.000083:	±		±01000.0-1000
2FA: -0.000005:	±		±01000.0-1000
2FB: -0.000026:	±		±01000.0-1000
2FC: 0.000093:	±		±01000.0-1000
2FD: -0.000015:	±		±01000.0-1000
2FE: -0.000090:	±		±01000.0-1000
2FF: 0.000570:	±		±01000.0-1000
300: 0.000011:	±		±01000.0-1000
301: -0.000307:	±		±01000.0-1000
302: 0.000133:	±		±01000.0-1000
303: -0.000116:	±		±01000.0-1000
304: 0.000032:	±		±01000.0-1000

0A-282075

270365-99

305: -0.000166: * | :510000.0- :005
 306: 0.000148: * | :040000.0- :005
 307: -0.000214: * | :020000.0- :005
 308: 0.000217: * | :001000.0- :005
 309: -0.000323: * | :001000.0- :005
 30A: 0.000205: * | :020000.0- :005
 30B: -0.000125: * | :021000.0- :005
 30C: 0.000172: * | :030000.0- :005
 30D: -0.000224: * | :000000.0- :005
 30E: 0.000131: * | :000000.0- :005
 30F: -0.000248: * | :001000.0- :005
 310: 0.000148: * | :000000.0- :005
 311: -0.000333: * | :000000.0- :005
 312: 0.000249: * | :001000.0- :005
 313: -0.000134: * | :000000.0- :005
 314: 0.000035: * | :000000.0- :005
 315: -0.000244: * | :000000.0- :005
 316: 0.000141: * | :000000.0- :005
 317: -0.000051: * | :000000.0- :005
 318: 0.000094: * | :001000.0- :005
 319: -0.000189: * | :000000.0- :005
 31A: 0.000070: * | :000000.0- :005
 31B: -0.000012: * | :000000.0- :005
 31C: 0.000158: * | :000000.0- :005
 31D: -0.000233: * | :001000.0- :005
 31E: 0.000096: * | :000000.0- :005
 31F: -0.000286: * | :000000.0- :005
 320: 0.000173: * | :001000.0- :005
 321: -0.000229: * | :000000.0- :005
 322: 0.000137: * | :001000.0- :005
 323: -0.000112: * | :000000.0- :005
 324: 0.000086: * | :000000.0- :005
 325: -0.000205: * | :000000.0- :005
 326: 0.000183: * | :001000.0- :005
 327: -0.000116: * | :000000.0- :005
 328: -0.000054: * | :000000.0- :005
 329: -0.000096: * | :001000.0- :005
 32A: 0.000149: * | :000000.0- :005
 32B: -0.000114: * | :000000.0- :005
 32C: 0.001180: * | :000000.0- :005
 32D: -0.000156: * | :001000.0- :005
 32E: 0.000086: * | :001000.0- :005
 32F: -0.000081: * | :001000.0- :005
 330: 0.000352: * | :000000.0- :005
 331: -0.000245: * | :000000.0- :005
 332: 0.000125: * | :001000.0- :005
 333: -0.000064: * | :000000.0- :005
 334: 0.000270: * | :001000.0- :005
 335: -0.000240: * | :000000.0- :005
 336: 0.000116: * | :000000.0- :005
 337: -0.000001: * | :000000.0- :005
 338: 0.000027: * | :000000.0- :005
 339: -0.000190: * | :000000.0- :005
 33A: -0.000001: * | :000000.0- :005
 33B: 0.000048: * | :000000.0- :005
 33C: 0.000128: * | :000000.0- :005
 33D: -0.000332: * | :000000.0- :005
 33E: 0.000222: * | :001000.0- :005
 33F: -0.000265: * | :001000.0- :005
 340: 0.001727: * | :000000.0- :005

88-28075

270365-A0

DNL Error, SN = 4130 (Continued)

341: -0.000352: *100|0-10T
 342: 0.000117: 15100|*10T
 343: -0.000287: 12100|0-10T
 344: 0.000340: 10100|0-10T
 345: -0.000116: 10000|0-10T
 346: -0.000079: 10000|0-10T
 347: 0.000078: 10000|0-10T
 348: 0.000241: 10000|0-10T
 349: 0.000037: 10000|0-10T
 34A: -0.000181: 15000|0-10T
 34B: -0.000080: 10000|0-10T
 34C: 0.000285: 10000|0-10T
 34D: -0.000360: 10000|0-10T
 34E: 0.000180: 10000|0-10T
 34F: 0.000050: 10000|0-10T
 350: 0.000002: 10000|0-10T
 351: -0.000219: 10000|0-10T
 352: 0.000092: 10000|0-10T
 353: -0.000083: 10000|0-10T
 354: 0.000366: 10000|0-10T
 355: -0.000302: 10000|0-10T
 356: 0.000188: 10000|0-10T
 357: 0.000024: 10000|0-10T
 358: -0.000125: 10000|0-10T
 359: -0.000215: 10000|0-10T
 35A: 0.000044: 10000|0-10T
 35B: 0.000026: 10000|0-10T
 35C: 0.000018: 10000|0-10T
 35D: -0.000315: 10000|0-10T
 35E: 0.000278: 10000|0-10T
 35F: -0.000044: 10000|0-10T
 360: -0.000112: 10000|0-10T
 361: -0.000169: 10000|0-10T
 362: 0.000062: 10000|0-10T
 363: -0.000132: 10000|0-10T
 364: 0.000127: 10000|0-10T
 365: -0.000235: 10000|0-10T
 366: 0.000109: 10000|0-10T
 367: -0.000064: 10000|0-10T
 368: 0.000157: 10000|0-10T
 369: -0.000121: 10000|0-10T
 36A: 0.000027: 10000|0-10T
 36B: -0.000109: 10000|0-10T
 36C: -0.000004: 10000|0-10T
 36D: -0.000294: 10000|0-10T
 36E: 0.000231: 10000|0-10T
 36F: -0.000030: 10000|0-10T
 370: 0.000065: 10000|0-10T
 371: -0.000268: 10000|0-10T
 372: 0.000114: 10000|0-10T
 373: -0.000050: 10000|0-10T
 374: 0.000087: 10000|0-10T
 375: -0.000210: 10000|0-10T
 376: 0.000048: 10000|0-10T
 377: 0.000111: 10000|0-10T
 378: -0.000073: 10000|0-10T
 379: -0.000186: 10000|0-10T
 37A: 0.000135: 10000|0-10T
 37B: -0.000020: 10000|0-10T
 37C: 0.000020: 10000|0-10T

270365-A1

37D: -0.000158:	:58C0010- :13C
37E: -0.000127:	:571A010- :58C
37F: 0.000509:	:0855010- :13A
380: -0.000187:	:3A20010- :13C
381: -0.000301:	:A100010- :13C
382: 0.000244:	:0F00010- :13A
383: -0.000066:	:0700010- :13C
384: 0.000016:	:1320010- :13C
385: -0.000098:	:1E00010- :13C
386: 0.000121:	:1010010- :13C
387: -0.000090:	:0000010- :13C
388: -0.000072:	:0850010- :13C
389: -0.000226:	:0000010- :13C
38A: 0.000078:	:0010010- :13C
38B: -0.000107:	:0000010- :13C
38C: -0.000009:	:1000010- :13C
38D: -0.000202:	:0120010- :13C
38E: 0.000292:	:1000010- :13C
38F: -0.000294:	:0000010- :13C
390: 0.000298:	:0000010- :13C
391: -0.000572:	* :58C0010- :13C
392: 0.000375:	:0010010- :13C
393: -0.000192:	:1020010- :13C
394: 0.000265:	:0510010- :13C
395: -0.000245:	:0100010- :13C
396: -0.000019:	:1000010- :13C
397: 0.000076:	:1000010- :13C
398: 0.000104:	:1000010- :13C
399: -0.000252:	:0000010- :13C
39A: 0.000148:	:0000010- :13C
39B: -0.000075:	:1000010- :13C
39C: 0.000243:	:1000010- :13C
39D: -0.000288:	:0000010- :13C
39E: 0.000147:	:1000010- :13C
39F: -0.000093:	:0010010- :13C
3A0: 0.000151:	:0310010- :13C
3A1: -0.000219:	:0000010- :13C
3A2: 0.000074:	:0010010- :13C
3A3: -0.000066:	:1000010- :13C
3A4: 0.000245:	:1000010- :13C
3A5: -0.000101:	:1000010- :13C
3A6: -0.000114:	:1000010- :13C
3A7: 0.000038:	:0010010- :13C
3A8: 0.000047:	:1000010- :13C
3A9: -0.000286:	:0000010- :13C
3AA: 0.000150:	:1000010- :13C
3AB: -0.000039:	:0000010- :13C
3AC: 0.000052:	:0000010- :13C
3AD: -0.000079:	:0000010- :13C
3AE: 0.000003:	:0010010- :13C
3AF: 0.000036:	:0000010- :13C
3B0: -0.000067:	:1000010- :13C
3B1: -0.000262:	:0010010- :13C
3B2: 0.000178:	:0000010- :13C
3B3: -0.000095:	:1000010- :13C
3B4: 0.000242:	:0000010- :13C
3B5: -0.000344:	:0000010- :13C
3B6: 0.000216:	:0010010- :13C
3B7: 0.000004:	:0000010- :13C
3B8: -0.000152:	:0000010- :13C

TA-880001

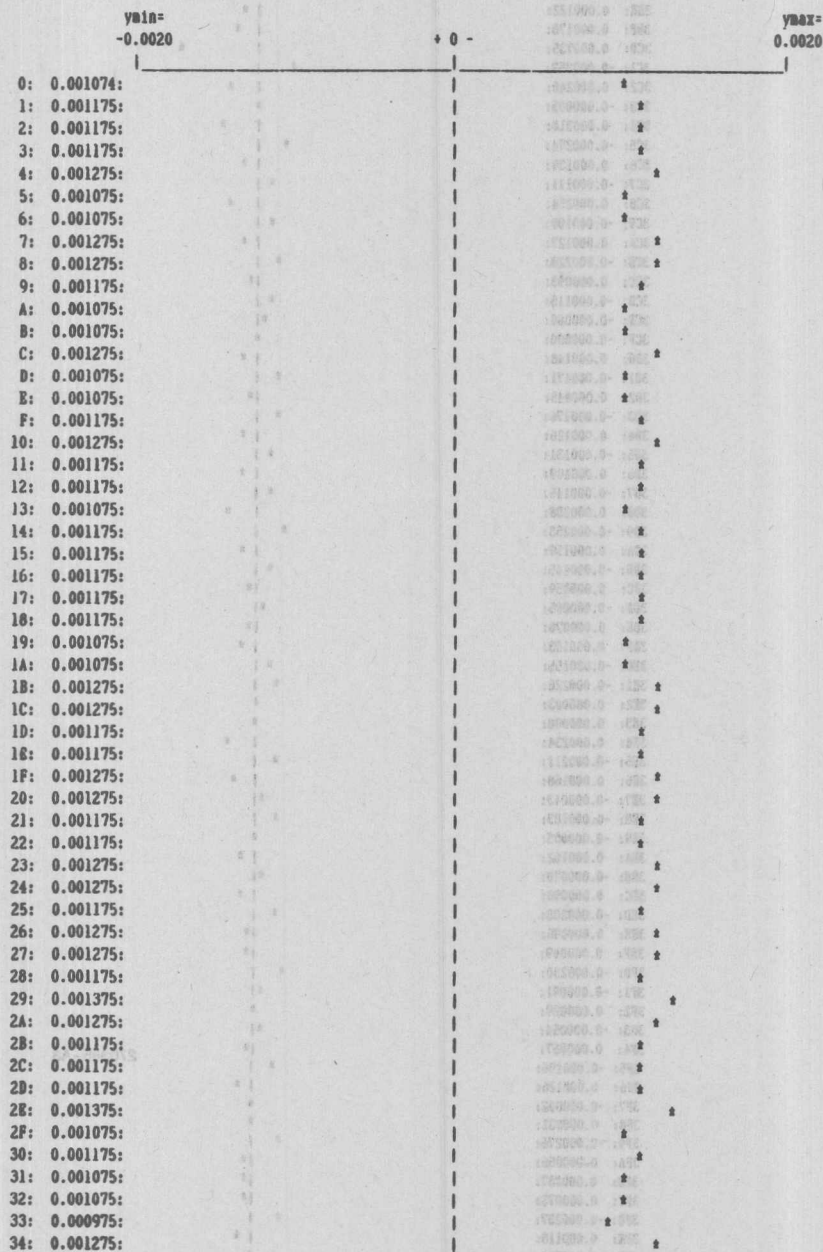
270365-A2

DNL Error, SN = 4130 (Continued)

3B9: -0.000106:	3B9: -0.000106:	3B9: -0.000106:	3B9: -0.000106:
3BA: 0.000187:	3BA: 0.000187:	3BA: 0.000187:	3BA: 0.000187:
3BB: -0.000069:	3BB: -0.000069:	3BB: -0.000069:	3BB: -0.000069:
3BC: 0.000075:	3BC: 0.000075:	3BC: 0.000075:	3BC: 0.000075:
3BD: -0.000028:	3BD: -0.000028:	3BD: -0.000028:	3BD: -0.000028:
3BE: 0.000122:	3BE: 0.000122:	3BE: 0.000122:	3BE: 0.000122:
3BF: 0.000178:	3BF: 0.000178:	3BF: 0.000178:	3BF: 0.000178:
3C0: 0.000735:	3C0: 0.000735:	3C0: 0.000735:	3C0: 0.000735:
3C1: -0.000359:	3C1: -0.000359:	3C1: -0.000359:	3C1: -0.000359:
3C2: 0.000248:	3C2: 0.000248:	3C2: 0.000248:	3C2: 0.000248:
3C3: -0.000005:	3C3: -0.000005:	3C3: -0.000005:	3C3: -0.000005:
3C4: 0.000318:	3C4: 0.000318:	3C4: 0.000318:	3C4: 0.000318:
3C5: -0.000274:	3C5: -0.000274:	3C5: -0.000274:	3C5: -0.000274:
3C6: 0.000139:	3C6: 0.000139:	3C6: 0.000139:	3C6: 0.000139:
3C7: -0.000111:	3C7: -0.000111:	3C7: -0.000111:	3C7: -0.000111:
3C8: 0.000254:	3C8: 0.000254:	3C8: 0.000254:	3C8: 0.000254:
3C9: -0.000100:	3C9: -0.000100:	3C9: -0.000100:	3C9: -0.000100:
3CA: 0.000127:	3CA: 0.000127:	3CA: 0.000127:	3CA: 0.000127:
3CB: -0.000228:	3CB: -0.000228:	3CB: -0.000228:	3CB: -0.000228:
3CC: 0.000093:	3CC: 0.000093:	3CC: 0.000093:	3CC: 0.000093:
3CD: -0.000115:	3CD: -0.000115:	3CD: -0.000115:	3CD: -0.000115:
3CE: -0.000060:	3CE: -0.000060:	3CE: -0.000060:	3CE: -0.000060:
3CF: -0.000000:	3CF: -0.000000:	3CF: -0.000000:	3CF: -0.000000:
3D0: 0.000148:	3D0: 0.000148:	3D0: 0.000148:	3D0: 0.000148:
3D1: -0.000171:	3D1: -0.000171:	3D1: -0.000171:	3D1: -0.000171:
3D2: 0.000045:	3D2: 0.000045:	3D2: 0.000045:	3D2: 0.000045:
3D3: -0.000176:	3D3: -0.000176:	3D3: -0.000176:	3D3: -0.000176:
3D4: 0.000126:	3D4: 0.000126:	3D4: 0.000126:	3D4: 0.000126:
3D5: -0.000131:	3D5: -0.000131:	3D5: -0.000131:	3D5: -0.000131:
3D6: 0.000109:	3D6: 0.000109:	3D6: 0.000109:	3D6: 0.000109:
3D7: -0.000145:	3D7: -0.000145:	3D7: -0.000145:	3D7: -0.000145:
3D8: 0.000288:	3D8: 0.000288:	3D8: 0.000288:	3D8: 0.000288:
3D9: -0.000253:	3D9: -0.000253:	3D9: -0.000253:	3D9: -0.000253:
3DA: 0.000159:	3DA: 0.000159:	3DA: 0.000159:	3DA: 0.000159:
3DB: -0.000145:	3DB: -0.000145:	3DB: -0.000145:	3DB: -0.000145:
3DC: 0.000059:	3DC: 0.000059:	3DC: 0.000059:	3DC: 0.000059:
3DD: -0.000085:	3DD: -0.000085:	3DD: -0.000085:	3DD: -0.000085:
3DE: 0.000078:	3DE: 0.000078:	3DE: 0.000078:	3DE: 0.000078:
3DF: 0.000103:	3DF: 0.000103:	3DF: 0.000103:	3DF: 0.000103:
3E0: -0.000155:	3E0: -0.000155:	3E0: -0.000155:	3E0: -0.000155:
3E1: -0.000228:	3E1: -0.000228:	3E1: -0.000228:	3E1: -0.000228:
3E2: 0.000003:	3E2: 0.000003:	3E2: 0.000003:	3E2: 0.000003:
3E3: 0.000008:	3E3: 0.000008:	3E3: 0.000008:	3E3: 0.000008:
3E4: 0.000234:	3E4: 0.000234:	3E4: 0.000234:	3E4: 0.000234:
3E5: -0.000211:	3E5: -0.000211:	3E5: -0.000211:	3E5: -0.000211:
3E6: 0.000168:	3E6: 0.000168:	3E6: 0.000168:	3E6: 0.000168:
3E7: -0.000043:	3E7: -0.000043:	3E7: -0.000043:	3E7: -0.000043:
3E8: -0.000183:	3E8: -0.000183:	3E8: -0.000183:	3E8: -0.000183:
3E9: -0.000005:	3E9: -0.000005:	3E9: -0.000005:	3E9: -0.000005:
3EA: 0.000162:	3EA: 0.000162:	3EA: 0.000162:	3EA: 0.000162:
3EB: -0.000070:	3EB: -0.000070:	3EB: -0.000070:	3EB: -0.000070:
3EC: 0.000098:	3EC: 0.000098:	3EC: 0.000098:	3EC: 0.000098:
3ED: -0.000208:	3ED: -0.000208:	3ED: -0.000208:	3ED: -0.000208:
3EE: 0.000096:	3EE: 0.000096:	3EE: 0.000096:	3EE: 0.000096:
3EF: 0.000049:	3EF: 0.000049:	3EF: 0.000049:	3EF: 0.000049:
3F0: -0.000230:	3F0: -0.000230:	3F0: -0.000230:	3F0: -0.000230:
3F1: -0.000091:	3F1: -0.000091:	3F1: -0.000091:	3F1: -0.000091:
3F2: 0.000029:	3F2: 0.000029:	3F2: 0.000029:	3F2: 0.000029:
3F3: -0.000054:	3F3: -0.000054:	3F3: -0.000054:	3F3: -0.000054:
3F4: 0.000067:	3F4: 0.000067:	3F4: 0.000067:	3F4: 0.000067:
3F5: -0.000196:	3F5: -0.000196:	3F5: -0.000196:	3F5: -0.000196:
3F6: 0.000126:	3F6: 0.000126:	3F6: 0.000126:	3F6: 0.000126:
3F7: -0.000002:	3F7: -0.000002:	3F7: -0.000002:	3F7: -0.000002:
3F8: 0.000031:	3F8: 0.000031:	3F8: 0.000031:	3F8: 0.000031:
3F9: -0.000276:	3F9: -0.000276:	3F9: -0.000276:	3F9: -0.000276:
3FA: 0.000056:	3FA: 0.000056:	3FA: 0.000056:	3FA: 0.000056:
3FB: 0.000087:	3FB: 0.000087:	3FB: 0.000087:	3FB: 0.000087:
3FC: 0.000073:	3FC: 0.000073:	3FC: 0.000073:	3FC: 0.000073:
3FD: -0.000237:	3FD: -0.000237:	3FD: -0.000237:	3FD: -0.000237:
3FE: 0.000118:	3FE: 0.000118:	3FE: 0.000118:	3FE: 0.000118:
3FF: 0.000000:	3FF: 0.000000:	3FF: 0.000000:	3FF: 0.000000:

DNL Error, SN = 4130 (Continued)

DI Array, SN = 4130



270365-51

Repeatability Error, SN = 4130

35:	0.001175:		:2V1100.0	:15V
36:	0.001075:		:2V1100.0	:15V
37:	0.001275:		:2V1100.0	:15V
38:	0.001075:		:2V1100.0	:15V
39:	0.001275:		:2V1100.0	:15V
3A:	0.001275:		:2V1100.0	:15V
3B:	0.001175:		:2V1100.0	:15V
3C:	0.001175:		:2V1100.0	:15V
3D:	0.001175:		:2V1100.0	:15V
3E:	0.001175:		:2V1100.0	:15V
3F:	0.001375:		:2V1100.0	:15V
40:	0.001275:		:2V1100.0	:15V
41:	0.001275:		:2V1100.0	:15V
42:	0.001075:		:2V1100.0	:15V
43:	0.001075:		:2V1100.0	:15V
44:	0.001275:		:2V1100.0	:15V
45:	0.001175:		:2V1100.0	:15V
46:	0.001175:		:2V1100.0	:15V
47:	0.001075:		:2V1100.0	:15V
48:	0.001175:		:2V1100.0	:15V
49:	0.001175:		:2V1100.0	:15V
4A:	0.001175:		:2V1100.0	:15V
4B:	0.001275:		:2V1100.0	:15V
4C:	0.001375:		:2V1100.0	:15V
4D:	0.001075:		:2V1100.0	:15V
4E:	0.001375:		:2V1100.0	:15V
4F:	0.001075:		:2V1100.0	:15V
50:	0.001175:		:2V1100.0	:15V
51:	0.001175:		:2V1100.0	:15V
52:	0.001175:		:2V1100.0	:15V
53:	0.000975:		:2V1100.0	:15V
54:	0.000975:		:2V1100.0	:15V
55:	0.001075:		:2V1100.0	:15V
56:	0.001175:		:2V1100.0	:15V
57:	0.001275:		:2V1100.0	:15V
58:	0.001275:		:2V1100.0	:15V
59:	0.001175:		:2V1100.0	:15V
5A:	0.001075:		:2V1100.0	:15V
5B:	0.001075:		:2V1100.0	:15V
5C:	0.001275:		:2V1100.0	:15V
5D:	0.001075:		:2V1100.0	:15V
5E:	0.001175:		:2V1100.0	:15V
5F:	0.001175:		:2V1100.0	:15V
60:	0.000975:		:2V1100.0	:15V
61:	0.001075:		:2V1100.0	:15V
62:	0.001075:		:2V1100.0	:15V
63:	0.001275:		:2V1100.0	:15V
64:	0.001275:		:2V1100.0	:15V
65:	0.001075:		:2V1100.0	:15V
66:	0.001175:		:2V1100.0	:15V
67:	0.001375:		:2V1100.0	:15V
68:	0.001075:		:2V1100.0	:15V
69:	0.001075:		:2V1100.0	:15V
6A:	0.001075:		:2V1100.0	:15V
6B:	0.001175:		:2V1100.0	:15V
6C:	0.001175:		:2V1100.0	:15V
6D:	0.001175:		:2V1100.0	:15V
6E:	0.001175:		:2V1100.0	:15V
6F:	0.001075:		:2V1100.0	:15V
70:	0.001075:		:2V1100.0	:15V

22-550075

270365-52

Repeatability Error, SN = 4130 (Continued)

71: 0.001375:			:2V1100.0 ±
72: 0.001175:			:2V0100.0 ± 100
73: 0.001175:			:2V5100.0 ± 100
74: 0.001175:			:2V0100.0 ± 100
75: 0.001175:			:2V5100.0 ± 100
76: 0.001175:			:2V5100.0 ± 100
77: 0.001275:			:2V1100.0 ± 100
78: 0.001175:			:2V1100.0 ± 100
79: 0.001275:			:2V1100.0 ± 100
7A: 0.001375:			:2V1100.0 ±
7B: 0.001375:			:2V5100.0 ±
7C: 0.001375:			:2V5100.0 ±
7D: 0.001375:			:2V5100.0 ±
7E: 0.001175:			:2V0100.0 ± 100
7F: 0.001075:			:2V0100.0 ± 100
80: 0.001275:			:2V5100.0 ± 100
81: 0.001175:			:2V1100.0 ± 100
82: 0.001275:			:2V1100.0 ± 100
83: 0.001275:			:2V0100.0 ± 100
84: 0.001075:			:2V1100.0 ± 100
85: 0.001175:			:2V1100.0 ± 100
86: 0.001175:			:2V1100.0 ± 100
87: 0.001275:			:2V5100.0 ± 100
88: 0.001075:			:2V0100.0 ± 100
89: 0.001075:			:2V0100.0 ± 100
8A: 0.001075:			:2V0100.0 ± 100
8B: 0.001075:			:2V0100.0 ± 100
8C: 0.001075:			:2V1100.0 ± 100
8D: 0.001175:			:2V1100.0 ± 100
8E: 0.001075:			:2V1100.0 ± 100
8F: 0.001175:			:2V0000.0 ± 100
90: 0.001175:			:2V0000.0 ± 100
91: 0.001175:			:2V0100.0 ± 100
92: 0.001275:			:2V1100.0 ± 100
93: 0.001175:			:2V5100.0 ± 100
94: 0.001075:			:2V5100.0 ± 100
95: 0.001175:			:2V1100.0 ± 100
96: 0.001275:			:2V0100.0 ± 100
97: 0.001075:			:2V0100.0 ± 100
98: 0.001175:			:2V1100.0 ± 100
99: 0.001175:			:2V0100.0 ± 100
9A: 0.001275:			:2V1100.0 ± 100
9B: 0.001175:			:2V1100.0 ± 100
9C: 0.001175:			:2V0000.0 ± 100
9D: 0.001275:			:2V0100.0 ± 100
9E: 0.001275:			:2V0100.0 ± 100
9F: 0.001175:			:2V5100.0 ± 100
A0: 0.001075:			:2V5100.0 ± 100
A1: 0.001175:			:2V0100.0 ± 100
A2: 0.001075:			:2V1100.0 ± 100
A3: 0.001275:			:2V5100.0 ± 100
A4: 0.001075:			:2V0100.0 ± 100
A5: 0.001175:			:2V0100.0 ± 100
A6: 0.001275:			:2V0100.0 ± 100
A7: 0.001375:			:2V1100.0 ±
A8: 0.001375:			:2V1100.0 ±
A9: 0.001075:			:2V1100.0 ± 100
AA: 0.001175:			:2V1100.0 ± 100
AB: 0.001075:			:2V0100.0 ± 100
AC: 0.001075:			:2V0100.0 ± 100

22-296052

270365-53

Repeatability Error, SN = 4130 (Continued)

AD: 0.001075:		1270100.0 * 181
AE: 0.001175:		1271100.0 * 182
AF: 0.001075:		1271100.0 * 183
BD: 0.001075:		1270100.0 * 184
B1: 0.001175:		1270100.0 * 185
B2: 0.001275:		1271100.0 * 186
B3: 0.001275:		1271100.0 * 187
B4: 0.000975:		1271100.0 * 188
B5: 0.001175:		1271100.0 * 189
B6: 0.001075:		1271100.0 * 190
B7: 0.001175:		1271100.0 * 191
B8: 0.001275:		1271100.0 * 192
B9: 0.001275:		1271100.0 * 193
BA: 0.001175:		1270100.0 * 194
BB: 0.001175:		1270100.0 * 195
BC: 0.001075:		1270100.0 * 196
BD: 0.001175:		1270100.0 * 197
BE: 0.001175:		1270100.0 * 198
BF: 0.001175:		1271100.0 * 199
C0: 0.001275:		1271100.0 * 200
C1: 0.001275:		1271100.0 * 201
C2: 0.001075:		1271100.0 * 202
C3: 0.000975:		1271100.0 * 203
C4: 0.001175:		1270100.0 * 204
C5: 0.001175:		1271100.0 * 205
C6: 0.001175:		1271100.0 * 206
C7: 0.001275:		1270100.0 * 207
C8: 0.001175:		1270100.0 * 208
C9: 0.001175:		1271100.0 * 209
CA: 0.001075:		1271100.0 * 210
CB: 0.001375:		1271100.0 * 211
CC: 0.001275:		1271100.0 * 212
CD: 0.001275:		1270100.0 * 213
CE: 0.001175:		1271100.0 * 214
CF: 0.001275:		1270100.0 * 215
D0: 0.001175:		1270100.0 * 216
D1: 0.001075:		1271100.0 * 217
D2: 0.001275:		1271100.0 * 218
D3: 0.001275:		1271100.0 * 219
D4: 0.001175:		1271100.0 * 220
D5: 0.001175:		1270100.0 * 221
D6: 0.001075:		1270100.0 * 222
D7: 0.001275:		1271100.0 * 223
D8: 0.001175:		1271100.0 * 224
D9: 0.001275:		1271100.0 * 225
DA: 0.001175:		1271100.0 * 226
DB: 0.001175:		1270100.0 * 227
DC: 0.001275:		1271100.0 * 228
DD: 0.001275:		1271100.0 * 229
DE: 0.001275:		1271100.0 * 230
DF: 0.001275:		1271100.0 * 231
E0: 0.001175:		1271100.0 * 232
E1: 0.000975:		1271100.0 * 233
E2: 0.001275:		1271100.0 * 234
E3: 0.001175:		1271100.0 * 235
E4: 0.001175:		1271100.0 * 236
E5: 0.001075:		1271100.0 * 237
E6: 0.001175:		1271100.0 * 238
E7: 0.001175:		1271100.0 * 239
E8: 0.001175:		1271100.0 * 240

22-280015

270365-54

E9:	0.001375:	1270100.0	1
EA:	0.001175:	1271100.0	1A
EB:	0.001175:	1270100.0	1A
EC:	0.001075:	1270100.0	1B
ED:	0.001375:	1271100.0	1
EE:	0.001375:	1270100.0	1
EF:	0.001275:	1270100.0	1B
FO:	0.001175:	1270000.0	1B
F1:	0.001175:	1271100.0	1B
F2:	0.001175:	1270100.0	1B
F3:	0.001275:	1271100.0	1B
F4:	0.001175:	1270100.0	1B
F5:	0.001275:	1270100.0	1B
F6:	0.001075:	1270100.0	1B
F7:	0.001375:	1271100.0	1
F8:	0.001075:	1270100.0	1B
F9:	0.001375:	1271100.0	1B
FA:	0.001275:	1271100.0	1B
FB:	0.001175:	1271100.0	1B
FC:	0.001275:	1270100.0	1B
FD:	0.001275:	1270100.0	1B
FE:	0.001275:	1270100.0	1B
FF:	0.001275:	1270000.0	1B
100:	0.001075:	1271100.0	1B
101:	0.001175:	1271100.0	1B
102:	0.001275:	1271100.0	1B
103:	0.001075:	1270100.0	1B
104:	0.001075:	1271100.0	1B
105:	0.001175:	1271100.0	1B
106:	0.001175:	1270100.0	1B
107:	0.001175:	1270100.0	1B
108:	0.001275:	1270100.0	1B
109:	0.001375:	1270100.0	1B
10A:	0.001275:	1271100.0	1B
10B:	0.001075:	1270100.0	1B
10C:	0.001075:	1271100.0	1B
10D:	0.001175:	1270100.0	1B
10E:	0.001175:	1270100.0	1B
10F:	0.001275:	1270100.0	1B
110:	0.001275:	1271100.0	1B
111:	0.001075:	1271100.0	1B
112:	0.001075:	1270100.0	1B
113:	0.001175:	1270100.0	1B
114:	0.001175:	1271100.0	1B
115:	0.001175:	1270100.0	1B
116:	0.001175:	1271100.0	1B
117:	0.001375:	1271100.0	1B
118:	0.001275:	1270100.0	1B
119:	0.001275:	1270100.0	1B
11A:	0.001175:	1270100.0	1B
11B:	0.001175:	1270100.0	1B
11C:	0.001175:	1271100.0	1B
11D:	0.001175:	1270000.0	1B
11E:	0.001175:	1270100.0	1B
11F:	0.001175:	1271100.0	1B
120:	0.001275:	1271100.0	1B
121:	0.001175:	1270100.0	1B
122:	0.001175:	1271100.0	1B
123:	0.001175:	1271100.0	1B
124:	0.001175:	1270100.0	1B

12-880013

270365-55

Repeatability Error, SN = 4130 (Continued)

125: 0.001175:
126: 0.001175:
127: 0.001275:
128: 0.001075:
129: 0.001175:
12A: 0.001275:
12B: 0.001175:
12C: 0.001175:
12D: 0.001275:
12E: 0.001075:
12F: 0.001275:
130: 0.001175:
131: 0.001175:
132: 0.001075:
133: 0.001075:
134: 0.001275:
135: 0.001275:
136: 0.001275:
137: 0.001075:
138: 0.001175:
139: 0.001275:
13A: 0.001175:
13B: 0.001275:
13C: 0.001175:
13D: 0.001175:
13E: 0.001275:
13F: 0.001075:
140: 0.001075:
141: 0.001075:
142: 0.001075:
143: 0.001075:
144: 0.001175:
145: 0.001275:
146: 0.001375:
147: 0.001375:
148: 0.001275:
149: 0.001275:
14A: 0.001275:
14B: 0.001275:
14C: 0.001375:
14D: 0.001375:
14E: 0.001175:
14F: 0.001175:
150: 0.001175:
151: 0.001275:
152: 0.001375:
153: 0.001275:
154: 0.001275:
155: 0.001175:
156: 0.001175:
157: 0.001275:
158: 0.001175:
15A: 0.001175:
15B: 0.001275:
15C: 0.001175:
15D: 0.001375:
15E: 0.001275:
15F: 0.001375:
160: 0.001375:

125: 0.001175:
126: 0.001175:
127: 0.001275:
128: 0.001075:
129: 0.001175:
12A: 0.001275:
12B: 0.001175:
12C: 0.001175:
12D: 0.001275:
12E: 0.001075:
12F: 0.001275:
130: 0.001175:
131: 0.001175:
132: 0.001075:
133: 0.001075:
134: 0.001275:
135: 0.001275:
136: 0.001275:
137: 0.001075:
138: 0.001175:
139: 0.001275:
13A: 0.001175:
13B: 0.001275:
13C: 0.001175:
13D: 0.001175:
13E: 0.001275:
13F: 0.001075:
140: 0.001075:
141: 0.001075:
142: 0.001075:
143: 0.001075:
144: 0.001175:
145: 0.001275:
146: 0.001375:
147: 0.001375:
148: 0.001275:
149: 0.001275:
14A: 0.001275:
14B: 0.001275:
14C: 0.001375:
14D: 0.001375:
14E: 0.001175:
14F: 0.001175:
150: 0.001175:
151: 0.001275:
152: 0.001375:
153: 0.001275:
154: 0.001275:
155: 0.001175:
156: 0.001175:
157: 0.001275:
158: 0.001175:
15A: 0.001175:
15B: 0.001275:
15C: 0.001175:
15D: 0.001375:
15E: 0.001275:
15F: 0.001375:
160: 0.001375:

270365-56

Repeatability Error, SN = 4130 (Continued)

270365-57

6-294

19D:	0.001275:	0.001275:
19E:	0.001275:	0.001275:
19F:	0.001275:	0.001275:
1A0:	0.001275:	0.001275:
1A1:	0.001275:	0.001275:
1A2:	0.001375:	0.001375:
1A3:	0.001475:	0.001475:
1A4:	0.001275:	0.001275:
1A5:	0.001075:	0.001075:
1A6:	0.001175:	0.001175:
1A7:	0.001275:	0.001275:
1A8:	0.001175:	0.001175:
1A9:	0.001375:	0.001375:
1AA:	0.001275:	0.001275:
1AB:	0.001275:	0.001275:
1AC:	0.001175:	0.001175:
1AD:	0.001175:	0.001175:
1AE:	0.001175:	0.001175:
1AF:	0.001175:	0.001175:
1B0:	0.001075:	0.001075:
1B1:	0.001375:	0.001375:
1B2:	0.000975:	0.000975:
1B3:	0.001175:	0.001175:
1B4:	0.001075:	0.001075:
1B5:	0.001375:	0.001375:
1B6:	0.001375:	0.001375:
1B7:	0.001175:	0.001175:
1B8:	0.001175:	0.001175:
1B9:	0.001375:	0.001375:
1BA:	0.001175:	0.001175:
1BB:	0.001475:	0.001475:
1BC:	0.001175:	0.001175:
1BD:	0.001275:	0.001275:
1BE:	0.001175:	0.001175:
1BF:	0.001175:	0.001175:
1C0:	0.001175:	0.001175:
1C1:	0.001175:	0.001175:
1C2:	0.001175:	0.001175:
1C3:	0.001275:	0.001275:
1C4:	0.001375:	0.001375:
1C5:	0.001275:	0.001275:
1C6:	0.001175:	0.001175:
1C7:	0.001175:	0.001175:
1C8:	0.001375:	0.001375:
1C9:	0.001175:	0.001175:
1CA:	0.001075:	0.001075:
1CB:	0.001075:	0.001075:
1CC:	0.001275:	0.001275:
1CD:	0.001375:	0.001375:
1CE:	0.001275:	0.001275:
1CF:	0.001075:	0.001075:
1D0:	0.001175:	0.001175:
1D1:	0.001275:	0.001275:
1D2:	0.001275:	0.001275:
1D3:	0.001275:	0.001275:
1D4:	0.001275:	0.001275:
1D5:	0.001275:	0.001275:
1D6:	0.001275:	0.001275:
1D7:	0.001175:	0.001175:
1D8:	0.001275:	0.001275:

270365-58

270365-59

6-296

215:	0.001275:		:271100.0 *	:125
216:	0.001275:		:273100.0 *	:025
217:	0.001275:		:275100.0 *	:625
218:	0.001275:		:271100.0 *	:145
219:	0.001375:		:275100.0 *	:125
21A:	0.001175:		:275100.0 *	:025
21B:	0.001275:		:275100.0 *	:725
21C:	0.001275:		:275100.0 *	:025
21D:	0.001275:		:275100.0 *	:025
21E:	0.001175:		:276100.0 *	:025
21F:	0.001175:		:271100.0 *	:025
220:	0.001375:		:276100.0 *	:125
221:	0.001275:		:275100.0 *	:025
222:	0.001375:		:275100.0 *	:125
223:	0.001375:		:271100.0 *	:125
224:	0.001175:		:275100.0 *	:005
225:	0.001375:		:271100.0 *	:125
226:	0.001175:		:271100.0 *	:505
227:	0.001375:		:275100.0 *	:125
228:	0.001175:		:275100.0 *	:125
229:	0.001275:		:275100.0 *	:125
22A:	0.001175:		:271100.0 *	:105
22B:	0.001275:		:275100.0 *	:125
22C:	0.001275:		:275100.0 *	:125
22D:	0.001175:		:275100.0 *	:125
22E:	0.001175:		:271100.0 *	:105
22F:	0.001075:		:271100.0 *	:105
230:	0.001275:		:275100.0 *	:105
231:	0.001175:		:275100.0 *	:105
232:	0.001175:		:275100.0 *	:105
233:	0.001275:		:275100.0 *	:125
234:	0.001275:		:271100.0 *	:015
235:	0.001375:		:275100.0 *	:125
236:	0.001375:		:271100.0 *	:125
237:	0.001275:		:275100.0 *	:015
238:	0.001275:		:275100.0 *	:015
239:	0.001175:		:271100.0 *	:015
23A:	0.001175:		:275100.0 *	:015
23B:	0.001175:		:275100.0 *	:015
23C:	0.001175:		:275100.0 *	:015
23D:	0.001375:		:275100.0 *	:125
23E:	0.001175:		:275100.0 *	:125
23F:	0.001275:		:275100.0 *	:015
240:	0.001275:		:275100.0 *	:015
241:	0.001275:		:271100.0 *	:015
242:	0.001275:		:275100.0 *	:015
243:	0.001275:		:275100.0 *	:015
244:	0.001275:		:275100.0 *	:005
245:	0.001375:		:275100.0 *	:125
246:	0.001075:		:275100.0 *	:125
247:	0.001175:		:275100.0 *	:125
248:	0.001175:		:275100.0 *	:105
249:	0.001375:		:275100.0 *	:125
24A:	0.001175:		:275100.0 *	:005
24B:	0.001275:		:275100.0 *	:125
24C:	0.001075:		:271100.0 *	:105
24D:	0.001175:		:275100.0 *	:005
24E:	0.001375:		:271100.0 *	:125
24F:	0.001375:		:275100.0 *	:125
250:	0.001275:		:271100.0 *	:005

10-385075

270365-60

251:	0.001175:		1275100.0	1215
252:	0.001275:		1275100.0	1215
253:	0.001275:		1275100.0	1215
254:	0.001175:		1275100.0	1215
255:	0.001375:		1275100.0	1215
256:	0.001275:		1275100.0	1215
257:	0.001275:		1275100.0	1215
258:	0.001275:		1275100.0	1215
259:	0.001175:		1275100.0	1215
25A:	0.001075:		1275100.0	1215
25B:	0.001175:		1275100.0	1215
25C:	0.001475:		1275100.0	1215
25D:	0.001275:		1275100.0	1215
25E:	0.001275:		1275100.0	1215
25F:	0.001175:		1275100.0	1215
260:	0.001275:		1275100.0	1215
261:	0.001175:		1275100.0	1215
262:	0.001175:		1275100.0	1215
263:	0.001275:		1275100.0	1215
264:	0.001275:		1275100.0	1215
265:	0.001275:		1275100.0	1215
266:	0.001175:		1275100.0	1215
267:	0.001375:		1275100.0	1215
268:	0.001275:		1275100.0	1215
269:	0.001275:		1275100.0	1215
26A:	0.001175:		1275100.0	1215
26B:	0.001175:		1275100.0	1215
26C:	0.001375:		1275100.0	1215
26D:	0.001375:		1275100.0	1215
26E:	0.001375:		1275100.0	1215
26F:	0.001475:		1275100.0	1215
270:	0.001175:		1275100.0	1215
271:	0.001375:		1275100.0	1215
272:	0.001175:		1275100.0	1215
273:	0.001075:		1275100.0	1215
274:	0.001275:		1275100.0	1215
275:	0.001175:		1275100.0	1215
276:	0.001275:		1275100.0	1215
277:	0.001375:		1275100.0	1215
278:	0.001375:		1275100.0	1215
279:	0.001375:		1275100.0	1215
27A:	0.001275:		1275100.0	1215
27B:	0.001275:		1275100.0	1215
27C:	0.001275:		1275100.0	1215
27D:	0.001175:		1275100.0	1215
27E:	0.001075:		1275100.0	1215
27F:	0.001275:		1275100.0	1215
280:	0.001275:		1275100.0	1215
281:	0.001375:		1275100.0	1215
282:	0.001275:		1275100.0	1215
283:	0.001275:		1275100.0	1215
284:	0.001275:		1275100.0	1215
285:	0.001375:		1275100.0	1215
286:	0.001275:		1275100.0	1215
287:	0.001375:		1275100.0	1215
288:	0.001175:		1275100.0	1215
289:	0.001275:		1275100.0	1215
28A:	0.001175:		1275100.0	1215
28B:	0.001375:		1275100.0	1215
28C:	0.001175:		1275100.0	1215

60-296012

270365-61

Repeatability Error, SN = 4130 (Continued)

[illegible]

270365-62

Repeatability Error, SN = 4130 (Continued)

2C9: 0.001375:				127E100.0	1285
2CA: 0.001175:				127E100.0	1283
2CB: 0.001275:				127E100.0	1283
2CC: 0.001275:				127E100.0	1283
2CD: 0.001475:				127E100.0	1284
2CE: 0.001275:				127E100.0	1285
2CF: 0.001175:				127E100.0	1285
2D0: 0.001175:				127E100.0	1285
2D1: 0.001175:				127E100.0	1285
2D2: 0.001275:				127E100.0	1285
2D3: 0.001375:				127E100.0	1285
2D4: 0.001175:				127E100.0	1285
2D5: 0.001175:				127E100.0	1285
2D6: 0.001275:				127E100.0	1285
2D7: 0.001375:				127E100.0	1285
2D8: 0.001275:				127E100.0	1285
2D9: 0.001275:				127E100.0	1285
2DA: 0.001475:				127E100.0	1285
2DB: 0.001475:				127E100.0	1285
2DC: 0.001375:				127E100.0	1285
2DD: 0.001375:				127E100.0	1285
2DE: 0.001375:				127E100.0	1285
2DF: 0.001375:				127E100.0	1285
2E0: 0.001175:				127E100.0	1285
2E1: 0.001375:				127E100.0	1285
2E2: 0.001275:				127E100.0	1285
2E3: 0.001175:				127E100.0	1285
2E4: 0.001175:				127E100.0	1285
2E5: 0.001275:				127E100.0	1285
2E6: 0.001275:				127E100.0	1285
2E7: 0.001375:				127E100.0	1285
2E8: 0.001175:				127E100.0	1285
2E9: 0.001275:				127E100.0	1285
2EA: 0.001275:				127E100.0	1285
2EB: 0.001175:				127E100.0	1285
2EC: 0.001375:				127E100.0	1285
2ED: 0.001275:				127E100.0	1285
2EE: 0.001275:				127E100.0	1285
2EF: 0.001175:				127E100.0	1285
2F0: 0.001275:				127E100.0	1285
2F1: 0.001375:				127E100.0	1285
2F2: 0.001375:				127E100.0	1285
2F3: 0.001275:				127E100.0	1285
2F4: 0.001275:				127E100.0	1285
2F5: 0.001375:				127E100.0	1285
2F6: 0.001475:				127E100.0	1285
2F7: 0.001375:				127E100.0	1285
2F8: 0.001375:				127E100.0	1285
2F9: 0.001475:				127E100.0	1285
2FA: 0.001275:				127E100.0	1285
2FB: 0.001175:				127E100.0	1285
2FC: 0.001275:				127E100.0	1285
2FD: 0.001275:				127E100.0	1285
2FE: 0.001275:				127E100.0	1285
2FF: 0.001275:				127E100.0	1285
300: 0.001075:				127E100.0	1285
301: 0.001275:				127E100.0	1285
302: 0.001375:				127E100.0	1285
303: 0.001275:				127E100.0	1285
304: 0.001175:				127E100.0	1285

SD-000075

270365-63

Repeatability Error, SN = 4130 (Continued)

305:	0.001475:	1202100.0	1202
306:	0.001275:	1202100.0	1202
307:	0.001375:	1202100.0	1202
308:	0.001375:	1202100.0	1202
309:	0.001275:	1202100.0	1202
30A:	0.001275:	1202100.0	1202
30B:	0.001375:	1202100.0	1202
30C:	0.001275:	1202100.0	1202
30D:	0.001475:	1202100.0	1202
30E:	0.001275:	1202100.0	1202
30F:	0.001375:	1202100.0	1202
310:	0.001175:	1202100.0	1202
311:	0.001175:	1202100.0	1202
312:	0.001275:	1202100.0	1202
313:	0.001275:	1202100.0	1202
314:	0.001375:	1202100.0	1202
315:	0.001275:	1202100.0	1202
316:	0.001275:	1202100.0	1202
317:	0.001275:	1202100.0	1202
318:	0.001175:	1202100.0	1202
319:	0.001375:	1202100.0	1202
31A:	0.001275:	1202100.0	1202
31B:	0.001075:	1202100.0	1202
31C:	0.001175:	1202100.0	1202
31D:	0.001375:	1202100.0	1202
31E:	0.001375:	1202100.0	1202
31F:	0.001375:	1202100.0	1202
320:	0.001375:	1202100.0	1202
321:	0.001375:	1202100.0	1202
322:	0.001375:	1202100.0	1202
323:	0.001275:	1202100.0	1202
324:	0.001174:	1202100.0	1202
325:	0.001575:	1202100.0	1202
326:	0.001275:	1202100.0	1202
327:	0.001475:	1202100.0	1202
328:	0.001174:	1202100.0	1202
329:	0.001375:	1202100.0	1202
32A:	0.001275:	1202100.0	1202
32B:	0.001275:	1202100.0	1202
32C:	0.001375:	1202100.0	1202
32D:	0.001375:	1202100.0	1202
32E:	0.001375:	1202100.0	1202
32F:	0.001375:	1202100.0	1202
330:	0.001174:	1202100.0	1202
331:	0.001174:	1202100.0	1202
332:	0.001475:	1202100.0	1202
333:	0.001275:	1202100.0	1202
334:	0.001275:	1202100.0	1202
335:	0.001575:	1202100.0	1202
336:	0.001475:	1202100.0	1202
337:	0.001174:	1202100.0	1202
338:	0.001275:	1202100.0	1202
339:	0.001275:	1202100.0	1202
33A:	0.001275:	1202100.0	1202
33B:	0.001275:	1202100.0	1202
33C:	0.001375:	1202100.0	1202
33D:	0.001375:	1202100.0	1202
33E:	0.001275:	1202100.0	1202
33F:	0.001275:	1202100.0	1202
340:	0.001174:	1202100.0	1202

20-000000

270365-64

Repeatability Error, SN = 4130 (Continued)

341:	0.001375:		1271100.0	*200
342:	0.001375:		1271100.0	*200
343:	0.001375:		1271100.0	*200
344:	0.001174:		1271100.0	*200
345:	0.001275:		1271100.0	*200
346:	0.001174:		1271100.0	*200
347:	0.001575:		1271100.0	*200
348:	0.001375:		1271100.0	*200
349:	0.001275:		1271100.0	*200
34A:	0.001174:		1271100.0	*200
34B:	0.001275:		1271100.0	*200
34C:	0.001275:		1271100.0	*200
34D:	0.001275:		1271100.0	*200
34E:	0.001275:		1271100.0	*200
34F:	0.001275:		1271100.0	*200
350:	0.001375:		1271100.0	*200
351:	0.001375:		1271100.0	*200
352:	0.001375:		1271100.0	*200
353:	0.001375:		1271100.0	*200
354:	0.001275:		1271100.0	*200
355:	0.001375:		1271100.0	*200
356:	0.001375:		1271100.0	*200
357:	0.001375:		1271100.0	*200
358:	0.001074:		1271100.0	*200
359:	0.001275:		1271100.0	*200
35A:	0.001074:		1271100.0	*200
35B:	0.001375:		1271100.0	*200
35C:	0.001375:		1271100.0	*200
35D:	0.001375:		1271100.0	*200
35E:	0.001275:		1271100.0	*200
35F:	0.001375:		1271100.0	*200
360:	0.001174:		1271100.0	*200
361:	0.001375:		1271100.0	*200
362:	0.001275:		1271100.0	*200
363:	0.001275:		1271100.0	*200
364:	0.001275:		1271100.0	*200
365:	0.001375:		1271100.0	*200
366:	0.001375:		1271100.0	*200
367:	0.001375:		1271100.0	*200
368:	0.001275:		1271100.0	*200
369:	0.001174:		1271100.0	*200
36A:	0.001275:		1271100.0	*200
36B:	0.001375:		1271100.0	*200
36C:	0.001375:		1271100.0	*200
36D:	0.001275:		1271100.0	*200
36E:	0.001275:		1271100.0	*200
36F:	0.001475:		1271100.0	*200
370:	0.001375:		1271100.0	*200
371:	0.001275:		1271100.0	*200
372:	0.001375:		1271100.0	*200
373:	0.001375:		1271100.0	*200
374:	0.001275:		1271100.0	*200
375:	0.001275:		1271100.0	*200
376:	0.001275:		1271100.0	*200
377:	0.001275:		1271100.0	*200
378:	0.001275:		1271100.0	*200
379:	0.001575:		1271100.0	*200
37A:	0.001475:		1271100.0	*200
37B:	0.001375:		1271100.0	*200
37C:	0.001275:		1271100.0	*200

48-088073

270365-65

Repeatability Error, SN = 4130 (Continued)

[illegible]

270365-66

Repeatability Error, SN = 4130 (Continued)

3B9: 0.001275:
 3BA: 0.001275:
 3BB: 0.001475:
 3BC: 0.001275:
 3BD: 0.001375:
 3BE: 0.001174:
 3BF: 0.001275:
 3C0: 0.001275:
 3C1: 0.001174:
 3C2: 0.001174:
 3C3: 0.001475:
 3C4: 0.001275:
 3C5: 0.001275:
 3C6: 0.001375:
 3C7: 0.001174:
 3C8: 0.001275:
 3C9: 0.001275:
 3CA: 0.001174:
 3CB: 0.001375:
 3CC: 0.001375:
 3CD: 0.001375:
 3CE: 0.001275:
 3CF: 0.001275:
 3D0: 0.001475:
 3D1: 0.001875:
 3D2: 0.001375:
 3D3: 0.001375:
 3D4: 0.001375:
 3D5: 0.001375:
 3D6: 0.001375:
 3D7: 0.001375:
 3D8: 0.001375:
 3D9: 0.001275:
 3DA: 0.001174:
 3DB: 0.001475:
 3DC: 0.001475:
 3DD: 0.001375:
 3DE: 0.001275:
 3DF: 0.001375:
 3E0: 0.001174:
 3E1: 0.001575:
 3E2: 0.001375:
 3E3: 0.001275:
 3E4: 0.001275:
 3E5: 0.001375:
 3E6: 0.001475:
 3E7: 0.001275:
 3E8: 0.001275:
 3E9: 0.001575:
 3EA: 0.001575:
 3EB: 0.001275:
 3EC: 0.001275:
 3ED: 0.001375:
 3EE: 0.001174:
 3EF: 0.001475:
 3F0: 0.001275:
 3F1: 0.001375:
 3F2: 0.001174:
 3F3: 0.001475:
 3F4: 0.001475:
 3F5: 0.001174:
 3F6: 0.001275:
 3F7: 0.001275:
 3F8: 0.001275:
 3F9: 0.001275:
 3FA: 0.001174:
 3FB: 0.001275:
 3FC: 0.001275:
 3FD: 0.001275:
 3FE: 0.001375:

270365-67
 270365-68

Repeatability Error, SN = 4130 (Continued)

Repeatability Error, SN = 4130 (Continued)

APPENDIX E BIBLIOGRAPHY

A/D Processing with Microcontrollers, Katausky, Horden, Smith

Apfel, R., et. al., "Signal-Processing Chips enrich telephone line- card Architecture". Electronics, May 5, 1982.

Analog Devices - Data-Acquisition Databook 1984, Volume 1

Blahut, Richard E., "Fast algorithms for digital signal processing", Addison Wesley Publishing Company, Inc., 1985.

Boyes, ed. - Syncro and Resolver Conversion, 1980

Brown, Robert Grover, "Introduction to random signal analysis and Kalman filtering". John Wiley & Sons, Inc., 1983.

Burr-Brown Application Note, Testing of Analog-to-Digital Converters

Burton and Dexter - Microprocessor Systems Handbook, 1977

Candy, J., et. al., "The Structure of Quantization Noise from Sigma-Delta Modulation", IEEE Transaction on Comm. Vol. Com. 29, No. 9, Sept. 1981.

Candy, J., et. al., "Using Triangularly Weighted Interpolation to Get 13-Bit PCM from a Sigma-Delta Modulator", IEEE Transaction on Comm., Nov. 1976.

Electronic Analog-to-Digital Converters, Seitzer, Pretzl, Handy

Handbook of Electronic Calculations, Chapter 15, Analog-Digital Conversion

Harris Analog and Telecom Data Book

IEEE 162

IEEE STD. 746-1984

Intel Application Note AP-124 - High-Speed Digital Servos for Motor Control Using the 2920/21 Signal Processor

Intel Application Note AP-125 - Designing Microcontroller Systems for Electrically Noisy Environments

Irwensen, J., "Calculated Quantization Noise of Single - Integration Delta Modulation Coders" BSTJ Sept. 1969.

ITT Digital 2000 VLSI Digital TV System, MAA 2300 Audio A/D Converter, Edition 1983/9.

MIL-M-38510/135 June 4, 1984

MIL-M-38510/135 May 6, 1985

Modern Electronic Circuits Reference Manual

NBS Staff Reports, May/June 1981 P.22/23

Sheingold, ed. - Analog-Digital Conversion Handbook, 1972

Sheingold, ed. - Analog-Digital Conversion Notes, 1977

Sheingold, ed. - Non-Linear Circuits Handbook, 1974

Sheingold, ed. - Transducer Interfacing Handbook, 1980

Steele, R., "Delta Modulation Systems", Pentech Press Limited, 1975.

Taylor, Fred U., "Digital filter design handbook", Marcel Dekker, Inc., 1983.

Terminology Related to the Perf of S/H, A/D, D/A Circuits, IEEE Transactions

Many applications have throughput time requirements on the order of a few hundred milliseconds, and don't require real-time image analysis.

A Single-Chip Image Processor

A.L. Pai and S.H. Lin, Arizona State University, and David P. Ryan, Intel Corporation

Most of the research efforts on image processing focus on expanding the complexity and dimension of image analysis. Unfortunately, this emphasis results in algorithms that are so computationally intensive that expensive special-purpose vector and pipeline processors are required to evaluate an image fast enough to be considered "real-time." Not all applications, however, have the burdensome requirements of true real-life image analysis. Specifically, applications that have image throughput time requirements of greater than a few hundred milliseconds can use a lower cost, general-purpose microprocessor-based system. Applications that have even slower frame rates are candidates for not only the use of lower cost CPUs, but also allow for replacement of video-rate flash A/D converters with slower, less expensive converters.

Addressing the most cost-sensitive applications, the design described herein uses Intel's 16-bit microcontroller to implement a single-chip image processor. The on-chip 10-bit A/D converter of the controller digitizes the image of a charge injection device (CID) camera, while the chip's 16-bit CPU executes a library of standard vision algorithms and reports the results by passing a few tokens over an on-chip universal asynchronous receiver-transmitter (UART).

SYSTEM OVERVIEW

A block diagram of the single-chip im-

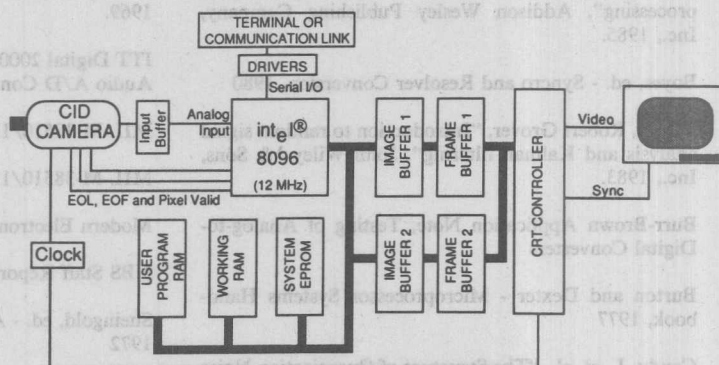


Figure 1. System block diagram.

age processor is shown in Figure 1. The image is acquired by the CID camera and input as an analog voltage to the 8096 where it is digitized and stored in one of two image buffers. The digital image is stored as an $N \times N$ matrix of 8-bit values corresponding to the gray level intensity at each picture element (pixel) as shown in Figure 2.

After an image resides in an image buffer, the 8096 can execute a number of standard image processing algorithms available as system monitor commands. These programs perform thresholding and filtering functions on the digital image, and can analyze objects found within the image. If the 8096 were attached to a host system instead of a terminal, custom pro-

grams could be downloaded to the user program RAM and executed.

To view the raw and processed images, a CRT controller is used to keep a video monitor updated with the images stored in the two frame buffer memories of Figure 1. The 8096 updates the frame buffers with the data in its image buffers depending upon commands given to the system.

► **Hardware.** The system is composed of a 128×128 CID camera and Intel's 8096 (with on-chip A/D) for image acquisition and analysis. A standard CRT controller was added for displaying raw and processed images as directed by the 8096. Driving the decision to use a 128×128 digital

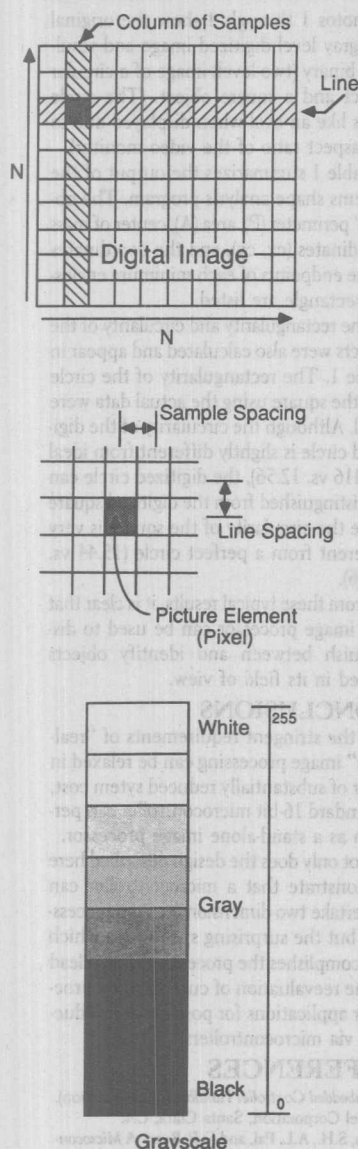


Figure 2. Representation of an $N \times N$ digital image.

image was the desire to store and operate upon two images simultaneously while minimizing memory requirements.

The image processing and communications software takes 5K of the 8K bytes allocated to the system monitor space, and would fit in the on-chip ROM space of an

8397 with 3K left. Two 32K x 8 SRAMs are used to provide space for two 16K byte image buffers, a 16K section of working RAM, and space for user-downloadable programs that are invoked by the monitor.

Two 16K byte frame-buffer memories are mapped to the same addresses as the corresponding image buffer used by the 8096. Normally, the frame buffers are mapped to the CRT controller to keep the video monitor updated. However, when the image stored in the image buffer is changed, the 8096 performs a frame-synchronized flyby block move to refresh the frame buffer (50 to 290 ms depending on whether frame synchronization is off or on).

To digitize an image, the 8096 monitors the end-of-line and end-of-frame signals from the CID camera and synchronizes the A/D conversion of the pixel data to a pixel valid signal from the camera. The analog output signal of the camera ranges from 0 to 1 V, corresponding to the gray level intensity at each pixel. This 1 V range is amplified to a 5 V range before being input to one of the eight A/D inputs of the 8096.

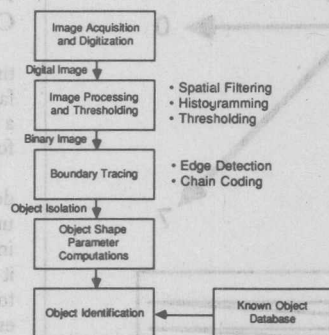


Figure 3. Object identification.

The 8096 converts the input voltage to a 10-bit digital representation in 22 μ s. Another 18 μ s are needed to store the pixel in the image buffer; update pointers, update a counter, and start another conversion. Therefore, the camera is clocked at a rate which results in a new pixel being output every 50 μ s.

Although the 8096 converts its analog input to 10 bits, the externally generated analog errors (such as buffer error and noise) led to the decision to use only 8 bits

of the result. This provides 256 gray levels, and greatly simplifies memory requirements.

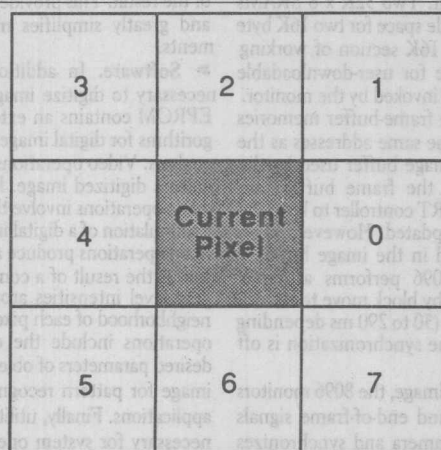
► **Software.** In addition to the code necessary to digitize images, the system EPROM contains an extensive set of algorithms for digital image acquisition and analysis. Video operations are used to acquire a digitized image. Point and arithmetic operations involve the pixel-by-pixel manipulation of a digital image. Neighborhood operations produce an output image that is the result of a combination of the gray level intensities around a specified neighborhood of each pixel. Measurement operations include the computation of desired parameters of objects located in an image for pattern recognition and other applications. Finally, utility operations are necessary for system operation.

The algorithms present in the system monitor can be used to identify desired objects in a digital image by following the approach shown in Figure 3. Once an image is digitized, it can be enhanced by the application of various image processing techniques including histogramming, thresholding, and spatial filtering to delineate the desired objects. The 8-directional chain code (Figure 4) can then be used to trace the boundaries of objects, and relevant object parameters can be determined and compared with those of a known object database to identify the unknown object.

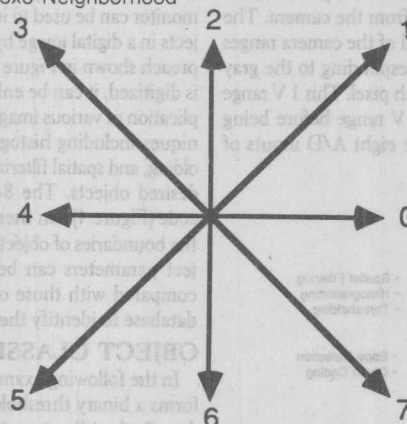
OBJECT CLASSIFICATION

In the following example, the 8096 performs a binary thresholding operation as described earlier to set the image background to pure white and the objects in the image to pure black. Then the 8096 searches the image for objects. When an object is found, the object boundary is traced and shape analysis is performed. Descriptive information about the object (or objects) is output over the on-chip UART of the 8096 to a terminal, or host computer. The controller, without consulting a host computer, can also be programmed to make the decision to accept or reject an object on a set of prescribed rules.

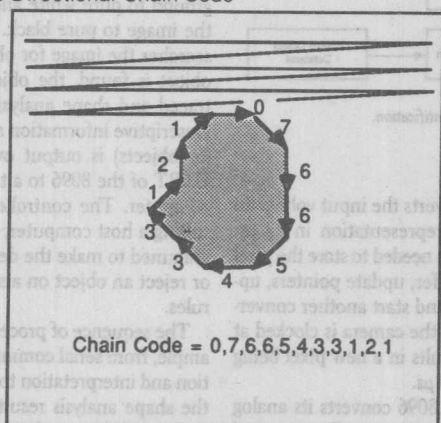
The sequence of processing for this example, from serial communication reception and interpretation to the reporting of the shape analysis results, takes approximately 1500 ms with an 8096 running at 12 MHz. The time will vary with the size and number of objects in the field of view.



(a) A 3x3 Neighborhood



(b) 8-Directional Chain-Code



Chain Code = 0,7,6,6,5,4,3,3,1,2,1

(c) Example Chain-Code of an Object

Figure 4. 8-directional boundary chain code.

Photos 1 through 4 show the original 256 gray level digitized image and resultant binary (two-level) image of a circular object and a square object. (The circle looks like an oval when displayed due to the aspect ratio of the video monitor).

Table 1 summarizes the output of the systems shape analysis program. The objects' perimeter (P), area (A), center of mass coordinates (cx, cy), and the coordinates of the endpoints of each minimum enclosing rectangle are listed.

The rectangularity and circularity of the objects were also calculated and appear in Table 1. The rectangularity of the circle and the square using the actual data were ideal. Although the circularity of the digitized circle is slightly different from ideal (12.416 vs. 12.56), the digitized circle can be distinguished from the digitized square since the circularity of the square is very different from a perfect circle (15.44 vs. 12.56).

From these typical results, it is clear that this image processor can be used to distinguish between and identify objects placed in its field of view.

CONCLUSIONS

If the stringent requirements of "real-time" image processing can be relaxed in favor of substantially reduced system cost, a standard 16-bit microcontroller can perform as a stand-alone image processor.

Not only does the design described here demonstrate that a microcontroller can undertake two-dimensional image processing, but the surprising speed with which it accomplishes the processing should lead to the reevaluation of current microprocessor applications for possible cost reduction via microcontrollers.

REFERENCES

1. *Embedded Controller Handbook*, (latest edition). Intel Corporation, Santa Clara, CA.
2. Lin, S.H., A.L. Pai, and D.P. Ryan. *A Microcontroller Based Digital Image Processor*, Proceeding of the Second World Conference on Robotics Research, MS86-766, Society of Manufacturing Engineers, Dearborn, MI.
3. Cunningham, R. "Segmenting Binary Images," *Robotics Age*, Vol. 5, No. 2, July/August 1981.
4. Wilf, J.M. "Chain Code," *Robotics Age*, Vol. 3, No. 2, March/April 1981.
5. Gonzalez, R. and P. Wintz. *Digital Image Processing*, Second Edition, Addison-Wesley Publishing Company, NY, 1987.
6. Fu, S.U., R.C. Gonzalez, and C.S.G. Lee. *Robotics: Control, Sensing, Vision and Intelligence*, McGraw-Hill Book Company, NY 1987.

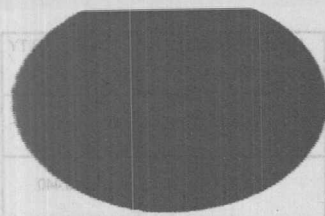


Photo 1. A digitized image of a circle.

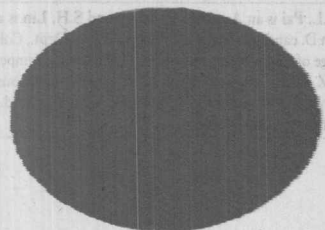


Photo 2. A thresholded binary (two-level) image of the same circle. The circle appears oval because of the aspect ratio of the video monitor.

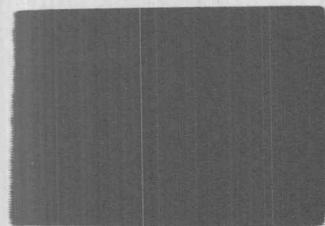


Photo 3. A digitized image of a square.

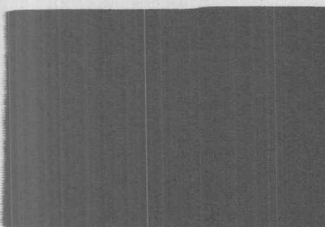


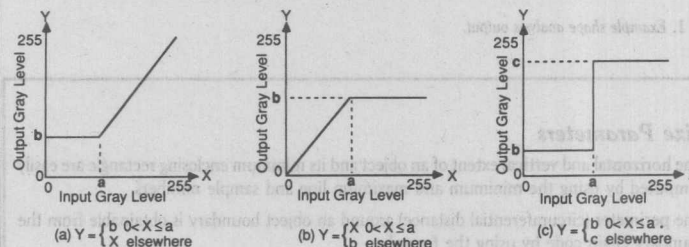
Photo 4. A thresholded binary image of the same square.

7. Castleman, K.R. *Digital Image Processing*, Prentice Hall International, Englewood Cliffs, NJ, 1985.
8. Baxes, G.A. *Digital Image Processing—A Practical Primer*, Prentice Hall International, Englewood Cliffs, NJ, 1984.

Image Processing Techniques

A histogram gives the distribution of all the gray levels in a digital image. The image histogram is used to select a desired threshold intensity level for separating an object from the background in the digital image.

A digital image can be thresholded using various threshold functions (three of which are shown in the figure) to yield an output image that contains a better definition of an object. For example, a binary (black and white) image is obtained by applying the two-level threshold function shown in (c).



In spatial filtering, the pixels adjacent to pixel (x, y) of image plane f are operated upon by the filter mask operation h . The resulting value of this spatial convolution is used to compute a replacement gray level intensity value at location (x, y) in the output image g . The following formula is used:

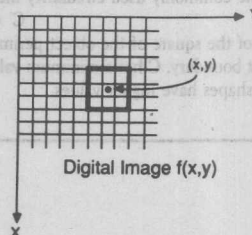
$$g(x, y) = h[f(x, y)] = [w_1 f(x-1, y-1) + w_2 f(x-1, y) + w_3 f(x-1, y+1) + w_4 f(x, y-1) + w_5 f(x, y) + w_6 f(x, y+1) + w_7 f(x+1, y-1) + w_8 f(x+1, y) + w_9 f(x+1, y+1)]$$

Various types of filter masks can be used to perform different digital image enhancement operations. A low-pass filter uses neighborhood averaging to "smooth" the digital image to remove noisy pixels. A high-pass filter accentuates noisy pixels. A high-pass filter accentuates the higher frequencies present in an image, thus "sharpening" its edges. Operators such as the Sobel masks can be used to compute the gradient at each point in an image, thus producing a gradient edge-detected image.

Using such filtering methods, the boundaries of objects in an image can be isolated, thus permitting the computation of useful object parameters for object identification and classification.

W_1 $(x-1, y-1)$	W_2 $(x-1, y)$	W_3 $(x-1, y+1)$
W_4 $(x, y-1)$	W_5 (x, y)	W_6 $(x, y+1)$
W_7 $(x+1, y-1)$	W_8 $(x+1, y)$	W_9 $(x+1, y+1)$

(a) A 3 x 3 Pixel Window with Spatial Mask coefficients W_i and Corresponding Image Pixel Locations



(b) A 3 x 3 Neighborhood Around Pixel (x, y)

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

(c) A Low-Pass Filter Mask

-1	1	-1
-1	9	-1
-1	-1	-1

(d) A High-Pass Filter Mask

-1	-2	-1
0	0	0
1	2	1

(e) Sobel Gradient Masks

EXAMPLE SHAPE ANALYSIS OUTPUT

OBJECT	PERIMETER P	AREA A	C.O.M. COORDINATES CX,CY	ENCLOSING RECTANGLE				RECTANGULARITY $R = A_O / A_R$	CIRCULARITY $C = P^2 / A$
				XMAX	XMIN	YMAX	YMIN		
CIRCLE	301	7297	(62,68)	111	15	116	21	0.785	12.416
SQUARE	328	6967	(55,73)	97	14	115	32	1.0	15.440

Table 1. Example shape analysis output.

Size Parameters

The horizontal and vertical extent of an object and its minimum enclosing rectangle are easily computed by using the minimum and maximum line and sample numbers.

The perimeter (circumferential distance) around an object boundary is obtainable from the boundary chain code by using the formula:

$$P = N_E + \sqrt{2} N_O$$

where N_E and N_O are the number of even and odd steps in the object boundary chain code.

The area of an object, which is a convenient measure of object size, is equal to the number of picture elements inside and including its boundary, multiplied by the area of a single pixel.

Shape Parameters

In addition to size parameters, shape parameters can be used to distinguish objects. Some shape parameters that are easily computed are described below.

The formula for computing the rectangularity R of an object is:

$$R = A_O / A_R$$

where A_O is the object area and A_R is the area of its minimum enclosing rectangle. R ranges from 0 to 1, with a value of 1.0 for rectangular objects, $\pi/4$ (0.785) for circular objects, and smaller values for slender, curved objects.

The aspect ratio, A , which is the ratio of width to length of the minimum enclosing rectangle of an object, is used to distinguish slender objects from roughly square or circular objects.

One of the commonly used circularity measures is:

$$C = P^2 / A$$

the ratio of the square of the object perimeter to its area, which reflects the complexity of the object boundary. C has a minimum value of 4π (12.56) for a circular object, while more complex shapes have higher values.

A.L. Pai is an Associate Professor and S.H. Lin is a Ph.D. candidate in the Computer Science Dept., College of Engineering, Arizona State University, Tempe, AZ 85281. David P. Ryan is a Senior Applications Engineer for Intel Corp., 5000 W. Chandler Blvd., Chandler, AZ 85226.

Reprinted from Sensors, June 1987
Copyright © 1987 by Helmers Publishing, Inc.
174 Concord St., Peterborough NH 03458
All Rights Reserved

SENSORS June 1987

1

1

THE MCS®-96 DIAGNOSTICS LIBRARY

Version X1.1

David Ryan
INTEL Corporation
October 1987

In the real time world of microcontroller applications, system failures can be dangerous and expensive. Preventing them, and understanding them when they occur, is very important. The source of a system upset can be a result of a software error, or a hardware failure. The application occurs as a result of a software error, or a hardware failure. The 8096 hardware provides methods of detecting and recovering from the transient noise failures, while the MCS-96 Diagnostics Library provides software routines that can help diagnose or detect a failure in system.

Unintentional recovery from noise induced failures is possible using the WATCHDOG TIMER. While the 8096-based system is functioning, the executing software periodically resets the WATCHDOG with a 12 MHz clock. If the WATCHDOG is not reset at least every 16 ms (12 MHz / 750), a system reset occurs. The two-byte code is a unique password which appears in the code map. This reduces the chance that an erroneous WATCHDOG test would occur after a system upset.

The 8096 RESET instruction provides another form of protection. Since the opcode for a RESET is 00H, protection against the 8096 executing unimplemented external memory is obtained by placing pull-ups on the system bus. The RESET opcode is also the value in erased EPROMs. Therefore, any attempt to execute non-existent memory or an erased EPROM location causes the 8096 to execute the RESET instruction. RESET causes the 8096 to reinitialize itself and provide an external pulse on the RESET pin to reinitialize the system.

Even with the protection afforded by the 8096, a system is rarely complete without checks for hardware failures. Both internal and external to the microcontroller. These checks are usually software routines that execute on power-up or periodically to verify that all parts of the system are present and function correctly. The tests generally execute standard check algorithms which are simply re-written in the host's assembly language.

To eliminate the need for every designer of an 8096-based system to write such tests, a collection of modular routines has been developed that any designer could easily use in his system (General Diagnostics). In addition, a set of 8096 interrupt service routines were developed for testing 8096 I/O units in a dedicated environment. (The Dynamic Stability Test). Both sets of programs are contained in the MCS-96 Diagnostics Library (DIAGLIB).

This library is a collection of software modules that provide a number of ready-made General Diagnostics and a specialized MCS-96 diagnostic known as the Dynamic Stability Test. The General Diagnostics implement frequently used standard test algorithms, while the Dynamic Stability Test exercises hardware specific to the 8096.

The library can be considered a software "tool box" from which modules are selected for a variety of run-time diagnostics or laboratory tests, for example:

- Include a few modules in other programs as a power-up test
- Use a memory module to create a map of external memory
- Use a few modules as a periodic system check
- Develop a simple stand-alone tester
- Build a custom test bed for burn-in, inspection or reliability tests

© Intel Corporation, 1987

October 1987
Order Number: 270083-002

1.0 INTRODUCTION

In the real time world of microcontroller applications, system failures can be dangerous, and expensive. Preventing them, and understanding them when they occur, is very important to the reliability of any design.

The sources of a system upset are varied. But in general, the failure of a well designed application occurs as a result of either some form of noise, or a hardware failure. The 8096 hardware provides methods of detecting and recovering from the transient noise failures, while the MCS®-96 Diagnostics Library supplies software routines that can help diagnose or detect a failure in system hardware.

Graceful recovery from noise induced failures is possible using the WATCHDOG TIMER. While the 8096-based system is functioning as desired, the executing software periodically resets the WATCHDOG with a special two-byte code. If the WATCHDOG is not reset at least every 16 ms (12 MHz system), a system reset occurs. The two-byte code is a unique password which appears nowhere in the opcode map. This reduces the chance that an erroneous WATCHDOG reset would occur after a system upset.

The 8096 RESET instruction provides another form of protection. Since the opcode for a RESET is 0ffH, protection against the 8096 executing unimplemented external memory is obtained by placing pull-ups on the system bus. The RESET opcode is also the value in erased EPROMs. Therefore, any attempt to execute non-existent memory or an erased EPROM location causes the 8096 to execute the RESET instruction. RESET causes the 8096 to reinitialize itself and provide an external pulse on the RESET pin to reinitialize the system.

Even with the protection afforded by the 8096, a system is rarely complete without checks for hardware failures, both internal and external to the microcontroller. These checks are usually software routines that execute on power-up or periodically to verify that all parts of the system are present and function correctly. The tests generally execute standard check algorithms which are simply re-written in the host's assembly language.

To eliminate the need for every designer of an 8096-based system to write such tests, a collection of modular routines has been developed that any designer could easily use in his system (**General Diagnostics**). In addition, a set of 8096 interrupt service routines was developed for testing 8096 I/O units in a dedicated environment (**The Dynamic Stability Test**). Both sets of programs are contained in the **MCS-96 Diagnostics Library (DIAG96.LIB)**.

This library is a collection of software modules that provide a number of ready-made **General Diagnostics** and a specialized MCS-96 diagnostic known as the **Dynamic Stability Test**. The **General Diagnostics** implement frequently used standard test algorithms, while the **Dynamic Stability Test** exercises hardware specific to the 8096.

The library can be considered a software "tool box" from which modules are selected for a variety of run-time diagnostics or laboratory tasks, for example:

- Include a few modules in other programs as a power-up test
- Use a memory module to create a map of external memory
- Use a few modules as a periodic system check
- Develop a simple stand-alone tester
- Build a custom test bed for burn-in, inspection or reliability tests
- Test new background code in an interrupt intensive environment

In addition to easing the development of a program that must perform standard diagnostics or system checks, the library can be a learning tool. Using the proven source code in the library, methods of interrupt management and on-chip peripheral handling can be reviewed to further understand how to use the 8096.

These tests were developed by the 8096 Applications group for experimental use with the 8096. With the programs in this library, the chip has been studied for its functional and asynchronous characteristics.

The **General Diagnostics** should be useful to almost anyone designing an 8096 application. The **Dynamic Stability Test** will be useful to those experimenting with the 8096 in a test environment. Figure 1 shows the modules in the **MCS-96 Diagnostics Library**.

1.1 General Diagnostics

The General Purpose Diagnostics consist of 24 programs providing System, ALU and Memory tests. Each of the tests can be called independently, and none require special hardware or impose application limiting constraints.

Two Collected Test programs are also provided so that all tests may be called at once. A third Collected Test program executes a selection of **General Diagnostics** that might be reasonable to include in a typical system power-up.

Section 3 provides a detailed description of the **General Diagnostics**.

1.2 Dynamic Stability Test

The **Dynamic Stability Test** is an integrated set of 11 programs that provide the interrupt service routines necessary to run all forms of MCS-96 I/O concurrently while a user written main task is executing. Virtually all of the chip is made to run simultaneously, with the I/O units responding to asynchronous external events.

Unlike the **General Diagnostics**, the **Dynamic Stability Test** modules must all be linked together, and must run in a specific external environment.

Section 4 provides a detailed description of the **Dynamic Stability Test**.

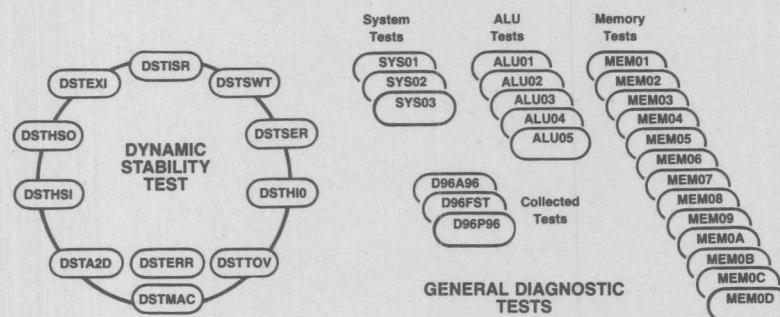


Figure 1. The MCS®-96 Diagnostic Library

1.3 How To Use This Manual

This publication is meant to be a guide for those using any of the programs in the **MCS-96 Diagnostics Library**. On a first pass the entire manual should be skimmed, with more attention paid to Section 2 and the overview sections of Sections 3 & 4. For the casual reader, the overview sections of each chapter should suffice.

Section 2 contains an overview of the general calling conventions to use any test in DIAG96.LIB. The section also describes DIAG96.LIB error reporting conventions and presents some warnings to heed when using this library.

Section 3 describes the classes of **General Diagnostics** and each test in detail.

Section 4 describes the concept of Dynamic Stability and its implementation on the 8096. The section also contains an overview of the tests performed, a description of the constraints placed upon the user-written background task, and detailed descriptions of each interrupt service routine.

The Appendices contain error code and command file descriptions, and of demonstration program listings. Source for the **MCS-96 Diagnostics Library** can be obtained from Insite User's Program Library at the address below. The Insite Catalog order number is AE-17.

Insite User's Program Library
INTEL Corporation DV2-24
2402 W. Beardsley Road
Phoenix, Arizona 85027

With the first-hand knowledge that many problems result from not being able to uncover information lodged in some dark corner of the user manual, information is repeated in the sections where it is pertinent.

2.0 USING THE LIBRARY

To simplify use of the diagnostics, the tests were developed in a modular fashion and collected in one linkable object file library (DIAG96.LIB). A modular program relies upon only the parameters sent at its invocation and employs standard parameter passing conventions to allow flexibility and uniformity of use. Collecting the modules into a library eliminates the tedium of listing twenty or thirty file names when performing a relocate/link on user developed code. When a program is linked to DIAG96.LIB, only those modules referenced in the user program are drawn from the library for inclusion in the output module.

Since PLM96 conventions were the ones chosen for this set of programs, the **General Diagnostics** are invoked by following the conventions for a PLM96 typed procedure. Parameters are placed on the STACK and the procedure activated via a function reference or explicit CALL. When the test is complete, test data is returned in the special register PLM\$REG. The **Dynamic Stability Test** is not PLM96 compatible.

The next section describes the format of the test data that is returned by the diagnostics. Following sections give an overview of how to use a **General Diagnostics** test, how to use **The Dynamic Stability Test**, and what restrictions to keep in mind while using the library.

2.1 Reporting Convention

All DIAG96.LIB tests use the PLM\$REG word locations 1CH and 1EH for returning condition codes to the calling program. Within DIAG96.LIB, these locations are the PUBLIC words EREG1 and EREG2. When a test concludes without finding an error, a zero is placed in the high byte of EREG1. If the high byte of EREG1 is non-zero, then some unexpected condition occurred. The low byte of EREG1 always contains the module number of the returning test, and EREG2 contains a detail code if an error was found. The complete listing of EREG1 code meanings and EREG2 meanings is in Appendices A & B.

All modules cease execution upon detection of the first error. The code describing which error was detected (EREG1) follows the format described in Table 1.

Table 1. Error Reporting Format

EREG1 = nnmx Hex			
where;	nn	= 00	if no error was found
		= 01, ..., 08H	if an error was found, nn is the error code
and;	mx	= 0xH	for Test = SYS0x; 1 ≤ x ≤ 03H
		= 1xH	ALU0x; 1 ≤ x ≤ 05H
		= 2xH	MEM0x; 1 ≤ x ≤ 0DH
		= 3xH	D96A96; x = 0
		= 4xH	DSTISR; x = 0
		= 5xH	DSTHSI; x = 1
		= 6xH	DSTHSO; x = 0
		= 7xH	DSTHSO; x = 0
		= 8xH	DSTHSO; x = 0
		= 9xH	DSTTOV; 0 ≤ x ≤ 1
		= 0AxH	DSTEXI; x = 0
		= 0BxH	DSTSER; x = 0
		= 0CxH	DSTA2D; x = 0
		= 0DxH	DSTSWT; 0 ≤ x ≤ 1
		= 0ExH	D96FST; x = 0
		= 0FxH	D96P96; x = 0

2.2 Using the General Diagnostics

The **General Diagnostics** provide a large set of system, ALU and memory tests that can be used in any combination, independent of system configuration or external circuitry. In addition to allowing for a wide flexibility in how a user's system is externally configured, the tests place minimal requirements on memory maps and interrupt environment.

Except where noted, all tests are interruptible, and maintain Program Status Word and Interrupt Mask integrity. The tests conform to PLM96 conventions, and require only run-time parameters to be passed for such specifics as memory test bounds and ALU test duration. To obtain access to the general diagnostics, the user should declare the needed module names **EXTERNAL** code segment symbols, and link to:

DIAG96.LIB

The tests are invoked in assembly language by placing the proper parameters on the STACK and CALLing the procedure. In PLM, the tests are run after a function reference is made with the appropriate parameters. The following is an example of an ASM96 call to a memory test:

```
PUSH    #4000h
PUSH    #5000h
CALL    MEM06
CMPB    EREG1+1,0
BNE     Error_Found
```

The diagnostic module called performs a complementary address test on the byte locations between 4000H and 5000H inclusive. If an error is found, the value returned in the word EREG1 will have a non-zero value as its high byte. Also in the case of an error, the MEM06 memory test will place the address of the error in location EREG2. The program D96A96, shown in Appendix D is a working ASM96 example that calls every **General Diagnostic Test**.

The same memory test could be called in a PLM96 program as follows:

```
Response = MEM06(4000h,5000h);
IF Error$Codes.Number > 255 THEN CALL Error$Found;
```

Since the diagnostics return two words in the PLM\$REG locations 1CH and 1EH, the function MEM06 would be a PROCEDURE of type LONG. Error\$Codes would have to be declared a STRUCTURE AT Response, with the word elements Number and Detail so that the error messages returned by the diagnostic can be stored. Number would contain the EREG1 value returned by the test, and Detail would contain EREG2. Response would have to be DECLARED a double word. The program D96P96, shown in Appendix D is a working PLM96 example that calls every **General Diagnostic**.

The action taken when an error is detected will depend upon the application. For example, the following Error_Found (or Error\$Found) routine would output the error codes to a printer or terminal:

```
Error_Found:      Error$Found: PROCEDURE;
PUSHF            DISABLE
PUSH    #Message_Ptr_A
CALL    Send_String
CALL output (.Message$Ptr$A,
            Error$Codes.Number);

PUSH    EREG1
CALL    Send_Hex_Word
CALL    Send_CR_LF
CALL output (.Message$Ptr$B,
            Error$Codes.Detail);

PUSH    #Message_Ptr_B
CALL    Send_String
Self: GOTO Self;
```

(Display continues on next page)


```

PUSH     SEND
CALL     Send_Hex_Word
CALL     Send_CR_LF
BR $
Message_Ptr_A:
DCB 27,'ERROR FOUND. Error Number = '
Message_Ptr_B:
DCB 22,'Error Detail Code is = '

```

In the Error_Found routine, it is assumed that the subroutines Send_String, Send_Hex_Word, and Send_CR_LF transmit appropriate ASCII codes given the parameters passed to them. Send_String is sent a pointer to a byte string in memory, the first byte of which is the character count. Send_Hex_Word converts the word put on the STACK into the correct four ASCII code bytes and appends the ASCII code for H. Send_CR_LF outputs the ASCII codes to cause a carriage return, followed by a line feed. The PLM routine output would perform similar operations.

To run the test, the user must supply a background task that calls an initialization routine (DSTR) with the specified parameters. After DSTR returns, the interrupt service routines will begin running. The background task can then perform any function that conforms to the constraints discussed in Section 4. If the user does not wish to write a special background task, one is provided in the module DSTR.

The following is an example CALL and a description of the parameters that must be passed to the initialization module (DSTR).

```

CALL     DSTR
PUSH     <RAM segment starting address>
PUSH     <RAM segment ending address>
PUSH     <RAM segment starting address>
PUSH     <RAM segment ending address>
PUSH     <random seed>
PUSH     <random test length>
PUSH     <argument for MultiplyDivide Core test>
PUSH     <argument for MultiplyDivide Core test>
PUSH     <bit pattern for memory test>

```

The RAM starting and ending addresses form a memory map for the memory tests that DSTR runs. The internal RAM is always tested. The random seed is the starting point for ALU tests that execute for as many number pairs as is specified in the random test length parameter. Argument1 and argument2 are the operands for a MultiplyDivide test. The bit pattern parameter is used during a memory test of the internal RAM and the memory segments specified.

Section 4 contains more detailed information on using the Dynamic Stability Test while the next section lists some general restrictions and assumptions that need to be understood to properly use any MCS-96 Diagnostic Library module.

2.3 Using the Dynamic Stability Test

The **Dynamic Stability Test** consists of a set of 8096 interrupt service routines that are designed to run while a user-supplied background task executes. The routines are located in the object file library DST96.LIB, which is contained in the master library DIAG96.LIB. To obtain access to the test, the user should invoke the batch file DSTRL.BAT with the background task file name and directory parameters. For example type:

```
DSTRL \SOURCE \BACK
```

Since the interrupt service routines test 8096 on-chip I/O devices, the part under test must reside in a specified hardware environment. Two such environments are available for use with the **Dynamic Stability Test**. The test may run in either a single chip mode, or a cross-coupled two chip mode. Figures 2 and 3 show the connections required for each configuration. In the single chip mode, output pins are connected to input pins on the same 8096. In the dual chip mode, output pins of one 8096 are connected to the input pins of the other (and vice versa).

To run the test, the user must supply a background task that CALLs an initialization routine (DSTISR) with the specified parameters. After DSTISR returns, the interrupt service routines will begin running. The background task can then perform any function that conforms to the constraints discussed in Section 4. If the user does not wish to write a special background task, one is provided in the module DSTUSR.

The following is an example CALL and a description of the parameters that must be passed to the initialization module (DSTISR).

```
PUSH    <RAM segment1 starting address>
PUSH    <RAM segment1 ending address>
PUSH    <RAM segment2 starting address>
PUSH    <RAM segment2 ending address>
PUSH    <random seed>
PUSH    <random test length>
PUSH    <argument1 for Multiply/Divide Core test>
PUSH    <argument2 for Multiply/Divide Core test>
PUSH    <bit pattern for memory test>
CALL    DSTISR
```

The RAM starting and ending addresses form a memory map for the memory tests that DSTISR runs. The internal RAM is always tested. The random seed is the starting point for ALU tests that execute for as many number pairs as is specified in the random test length parameter. Argument1 and argument2 are the operands for a Multiply/Divide test. The bit pattern parameter is used during a memory test of the internal RAM and the memory segments specified.

Section 4 contains more detailed information on using the **Dynamic Stability Test**, while the next section lists some general restrictions and assumptions that need to be understood to properly use any MCS-96 **Diagnostic Library** module.

2.4 Restrictions and Assumptions

Some general restrictions and assumptions need to be understood before any DIAG96.LIB programs can be successfully used.

- Pay close attention to the warnings about STACK location in the test modules you use. If you use any of the specialized internal register tests, make sure that the STACK is located externally. Do not partition a region of memory that contains your STACK in any memory test, unless you first move the STACK to an area you already tested.
- All **General Diagnostics** assume that the WATCHDOG TIMER is either being RESET by an interrupt service routine created by the user, or that it was never enabled. Only SYS02 ever locks out interrupts for a significant period of time. The amount of time they are locked out depends upon the parameters passed.
- **The Dynamic Stability Test** takes care of the WATCHDOG TIMER within its interrupt service routines. But, do not write to the WATCHDOG before CALLing the initialization subroutine.
- In any Dynamic Stability application, the user's Main Task should not lock out interrupts for more than a few instructions, as the CPU can get quite loaded down with interrupt requests that are very time dependent.

3.0 GENERAL DIAGNOSTICS

The 24 **General Diagnostics** included in DIAG96.LIB provide a good set of basic memory and ALU confidence tests that can be easily linked to application programs.

The **General Diagnostics** allow for a wide flexibility in how a user's system is configured with respect to memory maps and interrupt environment. Except where noted, all tests are interruptible, and maintain Program Status Word and interrupt mask integrity. The tests conform to PLM96 conventions, and require only run-time parameters to be passed for such specifics as memory test bounds and ALU test duration.

The tests are independent to allow specialized diagnostics to be developed as desired. Use just the quick power-up test (SYS02) to verify operation, or use the module that calls all **General Diagnostics** (D96A96) and let it run continuously for months. A module that performs the most common set of tests is also provided (D96FST).

The tests provided are of four classes: System Tests (SYSnn), ALU Tests (ALUnn), Memory Tests (MEMnn), and Collected Tests (D96xxx). To use any of the modules, from zero to ten parameters are PUSHed onto the STACK and the test is CALLED. Results are returned in the two word registers beginning at #1CH. The symbolic names for these locations (EREG1 and EREG2) are made PUBLIC if any DIAG96.LIB module is linked. They also may be referenced in PLM\$REG for PLM96 programs.

To obtain access to library modules, the user should declare the needed module names EXTERNAL code segment symbols, and link to:

DIAG96.LIB

The next few pages contain a brief overview of each of the four classes of tests. Then, the actions of each test are described in more detail.

System Tests

SYSnn

Common symbol definitions, storage reservations and two common routines are located in SYS01. A reference to any DIAG96.LIB module will cause SYS01 to be linked. SYS02 is meant to be called immediately after a RESET. It checks the special function register status and stack pointer, program status word and timer functionality. SYS03 is a simple program counter test. It does not test the complete range of the counter, and requires external RAM to execute.

SYS01: Common module
SYS02: RESET test
SYS03: Program counter exercise

ALU Tests

ALUnn

Five ALU modules are provided for checking ALU functionality. All report errors with a code in EREG1/EREG2.

Addition and subtraction are exercised in ALU01. A special eight-word add and subtract

is executed to test each adder bit with all possible combinations of a bit operation with and without carry-in.

Unsigned byte multiplication is verified by ALU02. This module simply executes all possible unsigned byte multiplications. Although not elegant, the test is effective. It takes six seconds.

A general test of the multiplication and division functions can be made with ALU03. The module executes all possible combinations of signed and unsigned, byte and word, two and three operand Multiplies and Divides using a specially selected table of numbers as operands.

ALU04 extends the ALU03 test by generating pseudo-random test pairs. The user program simply specifies a seed value for the random number generator, and the number of pairs to generate.

ALU05 is the core module for multiply/divide tests. Both ALU03 and ALU04 call ALU05. The user can also call ALU05 by passing a pair of test arguments. The module executes all possible combinations of signed and unsigned, byte and word, two and three operand Multiplies and Divides using the arguments passed as operands.

ALU01: Table-driven Addition/Subtraction
 ALU02: MULUB (all possible arguments)
 ALU03: Table-driven Multiply/Divide
 ALU04: Pseudo-random Multiply/Divide
 ALU05: Multiply/Divide core module

Memory Tests

MEMnn

The DIAG96.LIB MEMnn modules provide tests for register space, external RAM, and ROM. The algorithms used include: walking and galloping ones; walking and galloping zeros; checkerboard patterns; complementary addressing; and checksum verification.

The register tests are in MEM01-MEM05, and MEM0C. With the exception of MEM04, the register tests save the contents of all internal registers except PLM\$REG on the STACK before testing, and restore the data when done. If a faulty location is found, its address is reported. MEM04 is a utility which returns the number of bits set in a specified operand.

The external RAM tests are located in MEM06-MEM0A, and MEM0D. They all return a two-word code upon completion. The calling program must partition the RAM to be tested before calling an external RAM test.

Table 2. Memory Tests

Algorithm	Internal Registers	External RAM	ROM
Complementary Address	MEM01	MEM06	
Walking Ones		MEM07	
Walking Ones/Zeros	MEM02	MEM09	
Galloping Ones		MEM08	
Galloping Ones/Zeros	MEM03	MEM0A	
Bit Counter	MEM04		
Checkerboard Pattern	MEM05		
User Specified Pattern	MEM0C	MEM0D	
Checksum	MEM0B	MEM0B	MEM0B

Collected Tests

D96xxx

The D96xxx set of modules collects together all, or several, of the General Diagnostics and performs them according to the parameters passed. D96A96 is an ASM96 module that calls all tests. D96P96 is a PLM96 module that calls all tests. D96FST is an ASM96 module that calls a logical selection of tests.

D96A96: All tests / ASM96
 D96P96: All tests / PLM96
 D96FST: Selection of tests / ASM96

Memory Tests

MEMMn

The DIAG96 LIB MEMMn modules provide tasks for register space, external RAM, and ROM. The algorithms used include: walking and galloping ones; walking and galloping zeros; checksum patterns; complementary addressing; and checksum verification.

The register tests are in MEMM1-MEMM5, and MEMMOC. With the exception of MEMM1, the register tests save the contents of all internal registers except PLM96REG on the STACK before testing, and restore the data when done. If a faulty location is found, its address is reported. MEMM4 is a utility which returns the number of bits set in a specified operand.

The external RAM tests are located in MEMM6-MEMM9, and MEMMOC. They all return a two-word code upon completion. The calling program must partition the RAM to be tested before calling an external RAM test.

Table 2. Memory Tests

Algorithm	Internal Registers	External RAM	ROM
Complementary Address	MEMM1	MEMM6	
Walking Ones		MEMM7	
Walking Ones Zeros	MEMM2	MEMM8	
Galloping Ones		MEMM9	
Galloping Ones Zeros	MEMM3	MEMM4	
Bit Counter	MEMM5		
Checksum Pattern	MEMM6	MEMM5	
User Specified Pattern	MEMMOC	MEMMOC	
Checksum	MEMMOC	MEMMOC	MEMMOC

Common Symbols (SYS01)

Brief Description:

This module contains the global symbol declarations and five utilities used by the **General Diagnostics**.

Assembly Language Calling Sequence:

```

CALL    Get_Psw
      or
CALL    Put_Psw
      or
CALL    Get_Parms
      or
CALL    Stack_Ram
      or
CALL    Restore_Ram

Get_Psw Action:          Put_Psw Action:
USER_PSW := PSW         PSW := USER_PSW
EREG1    := 0
EREG2    := 0ffffh

Get_Parms Action:
PARM2 := Last Parameter
        put on the STACK
PARM1 := Next to last parameter
        put on the STACK
USER_PSW := PSW
EREG1    := 0ffffh
EREG2    := 0000h

Stack_Ram Action:       Restore_Ram Action:
PUSH 1aH;               Ptr := 0feh;
Ptr := 20H;              Do While Ptr > 1eh;
Do While Ptr < 100h;      POP [Ptr];
    PUSH [Ptr] +         Ptr := Ptr - 2;
End While;               End While;
                          POP 1aH;

```

Detailed Description:

A call to any **General Diagnostic** module will cause SYS01 to be linked. This module contains the definition of 4 words of memory used by every module to report errors and store temporary parameters. The STACK routines are used by the internal register tests to save and restore the data in the registers when called. It also INCLUDES an expanded 8096.INC file to provide the PUBLIC declarations of commonly used symbols for the special function registers and constants such as CR and LF.

Nearly all General Diagnostic modules use the routines in SYS01 to save the PSW when called, restore the PSW when returning control to the calling routine, save parameters from the STACK, and initialize the error registers.

System Power-up (SYS02)

Brief Description:

This test is a quick check of the Program Status Word, TIMER1, IOS0, IOS1 and the Interrupt Pending Register. It is meant to be called just after a RESET.

Assembly Language Calling Sequence:

```
CALL    SYS02
```

When Test Passes:

EREG1 := 0002h

EREG2 := 0000h

If Test Fails:

EREG1 := 0102h on unexpected IOS0 or IOS1 — EREG2 := IOS0 in low byte
IOS1 in high byte

EREG1 := 0202h if TIMER1 does not change — EREG2 := TIMER1

EREG1 := 0302h if Zero register failed — EREG2 := PSW at Failure

EREG1 := 0402h if PUSHF/POPF failed — EREG2 := erroneous value found

EREG1 := 0502h if Sticky bit failed — EREG2 := 3fffh if bit did not set
:= 0000h if bit did not clear

EREG1 := 0602h if Carry Flag failed — EREG2 := xxxxh

EREG1 := 0702h on an overflow flag error — EREG2 := 0002h if flags set wrong
:= xxxxh flags cleared wrong

EREG1 := 0802h if Int. Pending byte failed — EREG2 := offending Int. Pend. value

Detailed Description:

This module verifies that TIMER1 is changing, then attempts to change the value in the ZERO register. Then, a set of PUSHFs and POPFs is done with test values to verify correct action of these instructions. The carry, sticky and overflow bits in the program status word are then tested. Finally, the Interrupt Pending register bits are tested for their ability to be set and cleared. Any unexpected result is reported.

Any error found having to do with the PUSHF/POPF instructions or the PSW, including Interrupt Pending, will cause interrupts to be disabled before returning to the calling module.

Program Counter (SYS03)**Brief Description:**

This test writes code into a user selected partition of RAM and executes the code. Elapsed time and special registers are checked for correctness.

Assembly Language Calling Sequence:

```
PUSH    <start address>
PUSH    <end address>
CALL    SYS03
```

When Test Passes:

EREG1 := 0003h

EREG2 := 0000h

If Test Fails:

EREG1 := 0103h if test code returned early

EREG2 := Early time

EREG1 := 0203h if test code returned late

EREG2 := Late time

EREG1 := 0303h if count register is incorrect

EREG2 := erroneous counter value

Detailed Description:

This module accepts starting and ending addresses for an external RAM partition, adjusts the boundaries to be double word aligned, and writes three lines of code repeatedly into the partition. The code that is written increments a counter then executes two NOPs every 12 state times. The last byte written into the RAM partition is a RET opcode.

After the RAM partition is adjusted and the code written into the RAM, the test puts a return address on the STACK, stores TIMER1 and CALLs the first byte of the RAM. When the last byte of RAM is executed, program control returns to SYS03. TIMER1 is again stored. The test then compares the elapsed time to the expected elapsed time. The value remaining in the counter is also checked for correctness. Any deviations from expected are reported.

Caution: Since interrupts are locked-out while the code in RAM is executing, partitioning more than 4000h bytes of RAM for this test could cause a WATCHDOG TIMER overflow if the watchdog was started before SYS03 is called.

3.2 ALU Tests

Add/Subtract (ALU01)

Brief Description:

This routine adds then subtracts two carefully selected eight-word variables and verifies the results.

Assembly Language Calling Sequence:

```
CALL    ALU01
```

When Test Passes:

```
EREG1 := 0011h
```

```
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0111h on an addition error
```

```
:= 0211h on a subtraction error
```

```
:= 0311h on a flag error
```

```
EREG2 := offending argument on error
```

Detailed Description:

Two eight-word operands are added together and the results verified. Then, the operands are subtracted and verified. The operands were chosen to exercise every possible combination of two bits and a carry into each bit of the adder. Correctness of the result and the resultant flags is verified.

The operands are:

```
05555AAAAA5555AAAAFFFF0000AAAA5555H
```

```
+05555AAAAA5555FFFF00005555AAAAH
```

```
0AAAB555500000000FFFE0000FFFFFFFFH
```

```
05555AAAAA5555FFFF00005555AAAAH
```

```
-0AAAA5555AAAA55550000FFFF5555AAAAH
```

```
0AAAB555500000000FFFE0000FFFFFFFFH
```

Some versions of SIM96 do not pass this test.

Brief Description:

This module simply tests the MULUB instruction for all possible combinations of byte multipliers and multiplicands.

Assembly Language Calling Sequence:

CALL ALU02

When Test Passes:

EREG1 := 0012h
EREG2 := 0000h

If Test Fails:

EREG1 := 0112h on an error
EREG2 := multiplier/multiplicand

Detailed Description:

This test executes all possible combinations of operands into the MULUB instruction. Results are verified through a method of addition and subtraction as operands cycle. The status of PSW flags is not verified in this routine.

Multiply/Divide Table (ALU03)

Brief Description:

This module sends a specially constructed table of operands through the general Multiply/Divide Core test (ALU05).

Assembly Language Calling Sequence:

CALL ALU03

When Test Passes:

EREG1 := 0013h
EREG2 := 0000h

If Test Fails:

EREG1 := 0115h on a signed error
:= 0215h on an unsigned error
:= 0315h on a flag error
EREG2 := offending argument on error

Detailed Description:

This test sends a table of operands through the Multiply/Divide Core test. The 18 operands were selected to exercise all of the hardware multiply and divide control signals.

The operands are:

Arg.1,Arg.2		Arg.1,Arg.2	
1D99H,	0FFFFH	0FFFFH,	9D99H
9D99H,	5555H	5555H,	0E266H
0E266H,	0AAAAH	0AAAAH,	1D99H
1D99H,	5555H	5555H,	9D99H
9D99H,	0AAAAH	0AAAAH,	0E266H
0E266H,	0FFFFH	0FFFFH,	0063H
0063H,	0055H	0055H,	0066H
0066H,	00AAH	00AAH,	0063H
0063H,	00FFH		

Some versions of SIM96 will not pass this test.

Multiply/Divide Random (ALU04)**Brief Description:**

This module is a pseudo-random number generator that sends pairs of arguments to the Multiply/Divide Core test (ALU05).

Assembly Language Calling Sequence:

```
PUSH    <seed>
PUSH    <count>
CALL    ALU04
```

When Test Passes:

```
EREG1 := 0014h
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0115h on a signed error
      := 0215h on an unsigned error
      := 0315h on a flag error
EREG2 := offending argument on error
```

Detailed Description:

This module first executes the table driven Multiply/Divide test (ALU03). Then, if passed, pseudo-random argument pairs are generated and fed into the generalized Multiply/Divide Test (ALU05). The parameters passed to ALU04 set the random number seed, and the duration of the test.

There is no restriction on the values passed to the test. However, it must be noted that all possible combinations of signed and unsigned, byte and word, two and three operand Multiply/Divides are done at least twice for each pair of arguments sent to ALU05. Each such test takes from 1 to 5 milliseconds depending upon the arguments. Therefore, if large values for the count parameter are selected, the test will be long. For example, 1000h as a count will take about 12 seconds, depending upon the seed. NOTE: Some versions of SIM96 will not pass this test.

The formula used to generate the number pairs is as follows:

$$X(n+1) = [(0101h + 0001h) * X(n) + 0001h] \text{ MOD } 0ffffh$$

where X(0) = seed

Multiply/Divide Core (ALU05)**Brief Description:**

This test performs a Divide/re-Multiply sequence for all possible combinations of two or three operand, signed or unsigned, byte or word operations using the arguments passed to it as operands. The results are verified.

Assembly Language Calling Sequence:

```
PUSH    <argument1>
PUSH    <argument2>
CALL    ALU05
```

When Test Passes:

```
EREG1 := 0015h
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0115h on a signed error
      := 0215h on an unsigned error
      := 0315h on a flag error
```

```
EREG2 := offending argument on error
```

Detailed Description:

This module takes arguments from a calling program and performs upon them all possible combinations of byte or word, two or three operand, signed or unsigned multiplication and division. Argument2 is used to create the high and low words for a word Divide, and the low byte of Argument1 is used as the divisor in a byte Divide.

The test checks multiplication and division by first dividing one operand by the other, then multiplying the quotient by the divisor and adding the remainder. If the result is the original dividend, the operations were correct. However, the possibility of legitimate division overflows must also be considered.

The test first performs a division and checks flag status for correct indication of overflow conditions. If there has been an overflow, the dividend is right shifted by one, the expected result is updated, and the division is performed over. If a division by zero occurred, just the expected result is corrected and the test is continued.

After a division and overflow check/fixup is complete, a re-multiplication occurs and the result verified. Flag status is also verified. If the results are correct, the original operands are reloaded into the test operand registers and the next Divide/re-Multiply combination is begun.

All Divide/Multiply combinations are performed twice. Once with flags set upon entry, and once with flags clear upon entry.

CALLing ALU03 will run a specially selected table of operands through this test. CALLing ALU04 will run a pseudo-random string of operands through this test.

3.3 Memory Tests

Complementary Address (MEM01) (for registers)

Brief Description:

This module performs a complementary address test on the registers locations 1ah to 0ffh.

Assembly Language Calling Sequence:

```
CALL    MEM01
```

When Test Passes:

```
EREG1 := 0021h
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0121h
EREG2 := address of the error
```

Detailed Description:

This module performs a simple address and integrity test on register locations 1ah-0ffh. The algorithm stores the value NOT(ADDRESS) in the location pointed to by ADDRESS for the range, then loops through memory again to verify the contents.

Caution: If the STACK is partially internal, the STACK POINTER must be pointing at least 260 bytes into external RAM at the time MEM01 is called. The STACK cannot be entirely internal. The arithmetic flags in the PSW are undefined after execution of MEM01.

Walking Ones/Zeros (MEM02) (for registers)

Brief Description:

This module performs a Walking Ones and Zeros test on the internal registers 1ah-0ffh.

Assembly Language Calling Sequence:

```
CALL    MEM02
```

When Test Passes:

```
EREG1 := 0022h
```

```
EREG1 := 0000h
```

If Test Fails:

```
EREG1 := 0122h
```

```
EREG2 := address of the error
```

Detailed Description:

This module performs a Walking Ones and Zeros test on the internal registers.

The Walking Ones memory test first loads zero in all locations to be tested. Then, ones are placed in the first byte of memory, followed by a verification of all locations. Next, the first location is zeroed and ones are loaded into the second location. All memory is again verified. This process continues until all locations have been loaded with ones.

The Walking Zeros memory test works exactly like Walking Ones, except that a zero is "walked" through memory filled with ones, instead of ones being walked through a memory filled with zeros.

Caution: If the STACK is partially internal, the STACK POINTER must be pointing at least 260 bytes into external RAM at the time MEM02 is called. The STACK cannot be entirely internal. The arithmetic flags in the PSW are undefined after execution of MEM02.

Galloping Ones/Zeros (MEM03) (for registers)

Brief Description:

This module performs a Galloping Ones and Zeros test on the internal registers 1ah-0ffh.

Assembly Language Calling Sequence:

```
CALL    MEM03
```

When Test Passes:

```
EREG1 := 0023h
```

```
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0123h
```

```
EREG2 := address of the error
```

Detailed Description:

This module performs a Galloping Ones and Zeros test on internal registers.

The Galloping Ones algorithm tests memory by first loading zeros into all locations. Then ones are loaded into the first byte and all memory is verified. The verification is done by alternating reads to the first location and locations through all memory. Next, ones are placed in the second location without altering the first. Verification is again performed by alternating reads to the second location and the rest of memory. This process continues until all locations contain ones.

The Galloping Zeros test is similar to Galloping Ones, except that zeros slowly fill a memory filled with ones. In Galloping Ones, ones slowly fill a memory filled with zeros.

Caution: If the STACK is partially internal, the STACK POINTER must be pointing at least 260 bytes into external RAM at the time MEM03 is called. The STACK cannot be entirely internal. The arithmetic flags in the PSW are undefined after execution of MEM03.

Bits Set (MEM04)

Brief Description:

This module returns the number of bits set in the parameter passed to the routine.

Assembly Language Calling Sequence:

```
PUSH    test_value
CALL    MEM04
```

When All Bits Zero:

```
EREG1 := 0024h
```

```
EREG2 := 0000h
```

When One or More Bits Set:

```
EREG1 := 0124h
```

```
EREG2 := number of bits set
```

Detailed Description:

This module returns the number of bits that are set in the low byte of the parameter passed to the test. Any addressing mode may be used to put a value on the STACK, but the parameter on the STACK is treated as an immediate value.

Checkerboard Pattern (MEM05) (for registers)

Brief Description:

This module performs a Checkerboard Pattern test on the internal registers 1ah-Offh.

Assembly Language Calling Sequence:

```
CALL    MEM05
```

When Test Passes:

EREG1 := 0025h

EREG2 := 0000h

If Test Fails:

EREG1 := 0125h

EREG2 := address of the error

Detailed Description:

This module performs a checkerboard test on the internal registers. A checkerboard pattern of ones and zeros is written into the physical rows and columns of the 8096 register space. As the pattern is being written, it is repeatedly verified. After the entire pattern is in place, the memory is verified again, complemented, and re-verified.

Caution: If the STACK is partially internal, the STACK POINTER must be pointing at least 260 bytes into external RAM at the time MEM05 is called. The STACK cannot be entirely internal. The arithmetic flags in the PSW are undefined after execution of MEM05.

Complementary Address (MEM06)

Brief Description:

This module performs a complementary address test on the memory partitioned by user supplied pointers.

Assembly Language Calling Sequence:

```
PUSH    <start address>
PUSH    <end address>
CALL    MEM06
```

When Test Passes:

EREG1 := 0026h

EREG2 := 0000h

If Test Fails:

EREG1 := 0126h

EREG2 := offending address

Detailed Description:

This module performs a simple address and integrity test on RAM locations partitioned by the parameters passed. The algorithm stores the value NOT(ADDRESS) in the location pointed to by ADDRESS for the range, then loops through memory again to verify the contents.

Caution: Do not partition RAM that contains valid STACK elements.

Walking Ones (MEM07)**Brief Description:**

This module performs a Walking Ones Test on the memory partitioned by the user.

Assembly Language Calling Sequence:

```
PUSH    <start address>
PUSH    <end address>
CALL    MEM07
```

When Test Passes:

```
EREG1 := 0027h
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0127h
EREG2 := offending address
```

Detailed Description:

This module performs a Walking Ones test on the memory partitioned by the calling program. The Walking Ones memory test first loads zero in all locations to be tested. Then, ones are placed in the first byte of memory, followed by a verification of all locations. Next, the first location is zeroed and ones are loaded into the second location. All memory is again verified. This process continues until all locations have been loaded with ones.

Caution: Do not partition RAM that holds valid elements of the STACK. And, execution time increases non-linearly with memory partition widths.

Galloping Ones (MEM08)**Brief Description:**

This module performs a Galloping Ones test on memory partitioned by the calling program.

Assembly Language Calling Sequence:

```
PUSH    <start address>
PUSH    <end address>
CALL    MEM08
```

When Test Passes:

```
EREG1 := 0028h
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0128h
EREG2 := offending address
```

Detailed Description:

This module performs a Galloping Ones test on memory locations partitioned by the calling program.

The Galloping Ones algorithm tests memory by first loading zeros into all locations. Then ones are loaded into the first byte and all memory is verified. The verification is done by alternating reads to the first location and locations through all memory. Next, ones are placed in the second location without altering the first. Verification is again performed by alternating reads to the second location and the rest of memory. This process continues until all locations contain ones.

Caution: Do not partition locations that contain valid elements of the STACK. And, execution time increases non-linearly with memory partition widths.

Walking Ones/Zeros (MEM09)**Brief Description:**

This module performs a Walking Ones and Zeros test on the memory locations partitioned by the calling program.

Assembly Language Calling Sequence:

```
PUSH    <start address>
PUSH    <end address>
CALL    MEM09
```

When Test Passes:

EREG1 := 0029h

EREG2 := 0000h

If Test Fails:

EREG1 := 0129h

EREG2 := offending address

Detailed Description:

This module performs a Walking Ones and Zeros test on the memory partitioned by the calling program.

The Walking Ones memory test first loads zero in all locations to be tested. Then, ones are placed in the first byte of memory, followed by a verification of all locations. Next, the first location is zeroed and ones are loaded into the second location. All memory is again verified. This process continues until all locations have been loaded with ones.

The Walking Zeros memory test works exactly like Walking Ones, except that a zero is "walked" through memory filled with ones, instead of ones being walked through a memory filled with zeros.

Caution: Do not partition RAM that contains valid elements of the STACK. And, execution time increases non-linearly with memory partition widths.

Brief Description:

This module performs a Galloping Ones and Zeros test on the memory locations partitioned by the calling program.

Assembly Language Calling Sequence:

```
PUSH    <starting address>
PUSH    <ending address>
CALL    MEM0A
```

When Test Passes:**If Test Fails:**

EREG1 := 002Ah

EREG1 := 012Ah

EREG2 := 0000h

EREG2 := offending address

Detailed Description:

This module performs a Galloping Ones and Zeros test on memory partitioned by the calling program.

The Galloping Ones algorithm tests memory by first loading zeros into all locations. Then ones are loaded into the first byte and all memory is verified. The verification is done by alternating reads to the first location and locations through all memory. Next, ones are placed in the second location without altering the first. Verification is again performed by alternating reads to the second location and the rest of memory. This process continues until all locations contain ones.

The Galloping Zeros test is similar to Galloping Ones, except that zeros slowly fill a memory filled with ones. In Galloping Ones, ones slowly fill a memory filled with zeros.

Caution: Do not partition RAM that contains valid elements of the STACK. And, execution time increases non-linearly with memory partition widths.

Checksum (MEM0B)**Brief Description:**

This module calculates a 16 bit checksum for the memory partition specified by the calling program.

Assembly Language Calling Sequence:

```
PUSH    <starting address>
PUSH    <ending address>
CALL    MEM0B
```

Test Returns:

EREG1 := 012bh

EREG2 := 16-bit checksum

Detailed Description:

This module performs a 16-bit checksum on the region of memory partitioned by the calling program. RAM or ROM may be partitioned. The module is non-destructive to RAM.

(for registers)

Brief Description:

This module performs a Checkerboard Pattern test on the internal registers 1ah-0ffh with a user specified bit pattern.

Assembly Language Calling Sequence:

```
PUSH    <desired bit pattern>
CALL    MEM0C
```

When Test Passes:

EREG1 := 002Ch
EREG2 := 0000h

If Test Fails:

EREG1 := 012Ch
EREG2 := address of the error

Detailed Description:

This module performs a checkerboard test on the internal registers with the bit pattern specified by the calling program. The pattern is written into the physical rows and columns of the 8096 register space. As the pattern is being written, it is repeatedly verified. After the entire pattern is in place, the memory is verified again, complemented, and re-verified.

Caution: If the STACK is partially internal, the STACK POINTER must be pointing at least 260 bytes into external RAM at the time MEM0C is called. The STACK cannot be entirely internal. The arithmetic flags in the PSW are undefined after execution of MEM0C.

User Pattern (MEM0D)

Brief Description:

This module performs a Checkerboard Pattern test on a specified region of memory with a specified pattern of bits.

Assembly Language Calling Sequence:

```
PUSH    <starting address>
PUSH    <ending address>
PUSH    <bit pattern>
CALL    MEM0D
```

When Test Passes:

EREG1 := 002dh
EREG2 := 0000h

If Test Fails:

EREG1 := 012dh
EREG2 := offending address

Detailed Description:

This module performs a checkerboard test on a region of memory that is specified by the calling program using a bit pattern which is also specified. First, the pattern is written into memory. As the pattern is being written, it is repeatedly verified. After the entire pattern is in place, the memory is verified again, complemented, and re-verified.

Caution: Do not partition RAM that contains valid elements of the STACK.

3.4 Collected Tests Modules

ALL Tests in ASM96 (D96A96)

Brief Description:

This module causes every **General Diagnostics** test to execute.

Assembly Language Calling Sequence:

```

PUSH    <RAM segment1 starting address>
PUSH    <RAM segment1 ending address>
PUSH    <RAM segment2 starting address>
PUSH    <RAM segment2 ending address>
PUSH    <random seed>
PUSH    <random test length>
PUSH    <top of code address>
PUSH    <argument1 for Multiply/Divide Core test>
PUSH    <argument2 for Multiply/Divide Core test>
PUSH    <bit pattern for memory test>
CALL    D96A96

```

When Tests All Pass:

EREG1 := 0030h
EREG2 := code checksum

When a Test Fails:

EREG1 := test module error code
EREG2 := test module detail code

Detailed Description:

This module calls all **General Diagnostics** using the parameters passed by the calling program. The parameters needed by the test for proper execution specify two areas of external RAM for memory tests, the ending address of code to be checksummed, the seed and length of the random ALU test, two specific arguments to do the Multiply/Divide Core test, and a bit pattern for memory tests.

Execution speed of this test is highly dependent upon the memory partitions and the length requested for the random ALU test. For example, partitioning 1k and 8k regions of memory, and calling for 1000h random ALU tests, the test takes 3 hours to complete. Testing smaller regions of memory (i.e. 1k and 1k) can reduce test time to a few minutes.

Caution: An external STACK must be used with this test, and it must be in a part of memory outside that partitioned during the CALL.

ALL Tests in PLM96 (D96P96)**Brief Description:**

This module causes every **General Diagnostics** test module to execute.

PLM96 Calling Sequence:

D96P96(RAM segment1 starting address,
 RAM segment1 ending address,
 RAM segment2 starting address,
 RAM segment2 ending address,
 random seed, random test length,
 top of code address,
 argument1 for Multiply/Divide Core test,
 argument2 for Multiply/Divide Core test,
 bit pattern for memory tests);

When All Tests Pass:

PLMREG := 00F0h
 PLMREG + 2 := 16-bit checksum

When a Test Fails:

PLMREG := module error code
 PLMREG + 2 := module detail code

Detailed Description:

This module calls all **General Diagnostics** using the parameters passed during invocation. The parameters needed by the test for proper execution specify two areas of external RAM for memory tests, the ending address of code to be checksummed, the seed and length of the random ALU test, two specific arguments to do the Multiply/Divide Core test, and a bit pattern for memory tests.

Execution speed of this test is highly dependent upon the memory partitions and the length requested for the random ALU test. For example, partitioning 1k and 8k regions of memory, and calling for 1000h random ALU tests, the test takes 3 hours to complete. Testing smaller regions of memory (i.e. 1k and 1k) can reduce test time to a few minutes.

In his program, the user will have to DECLARE D96P96 an external procedure of the LONG type, with its parameters declared SLOW. The EREG1 and EREG2 values reported by library modules are placed in the long-word location at PLM\$REG.

The DECLARations in D96P96 show how any one General Diagnostic Module could be called from a PLM96 program. Each needed module needs to be DECLARED an external procedure of the LONG type.

Caution: An external STACK must be used with this test, and it must be in a part of memory outside that partitioned during the CALL.

Selected Tests in ASM (D96FST)**Brief Description:**

This is an ASM module that invokes a selected set of **General Diagnostic tests**.

Assembly Language Calling Sequence:

```

PUSH    <RAM segment1 starting address>
PUSH    <RAM segment1 ending address>
PUSH    <RAM segment2 starting address>
PUSH    <RAM segment2 ending address>
PUSH    <random seed>
PUSH    <random test length>
PUSH    <top of code address>
PUSH    <argument1 for Multiply/Divide Core test>
PUSH    <argument2 for Multiply/Divide Core test>
PUSH    <bit pattern for memory test>
CALL    D96FST

```

When Tests All Pass:

```

EREG1 := 00E0h
EREG2 := code checksum

```

When a Test Fails:

```

EREG1 := test module error code
EREG2 := test module detail code

```

Detailed Description:

This module calls the Power-up and Program Counter tests then all ALU tests. Then, Complementary Address, Galloping Ones/Zeros and Checkerboard tests are run on the internal registers. Finally, Complementary Address and specified pattern tests are done on external memory and the program is checksummed.

The parameters needed by the test for proper execution specify two areas of external RAM for memory tests, the ending address of code to be checksummed, the seed and length of the random ALU test, two specific arguments to do the Multiply/Divide Core test, and a bit pattern for memory tests.

Execution speed of this test is highly dependent upon the memory partitions and the length requested for the random ALU test. For example, partitioning 1k and 8k regions of memory, and calling for 1000h random ALU tests, the test takes about 20 seconds to complete. Testing smaller regions of memory (i.e. 1k and 1k) can reduce test time further.

Caution: An external STACK must be used with this test, and it must be in a part of memory outside that partitioned during the CALL.

4.0 THE DYNAMIC STABILITY TEST

The **Dynamic Stability Test** is a set of interrupt service routines designed to run over a user's background task in either one stand alone 8097, or two 8097s that are cross-coupled. In the stand alone mode, the chip's output pins are hooked to its input pins. In the dual chip mode, each controller's output pins are tied to the input pins of the other. The minimum configuration for each mode are shown in Figures 2 and 3. See Figure 11 for the circuit diagram of a board that can be jumpered for either configuration.

What is Dynamic Stability?

A "Dynamic Stability" test was developed to enable testing of the 8097 in an asynchronous environment. In the one chip mode, HSO events are synchronized with the HSI

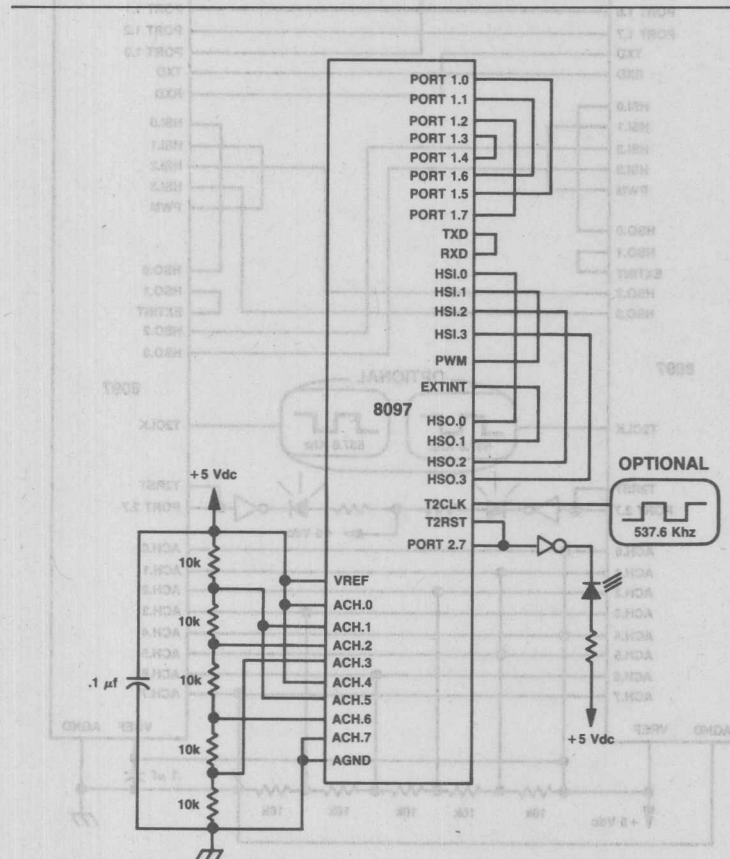


Figure 2. 8097 Strapback Configuration Single Chip Mode

event capture logic. However, in the cross-coupled mode, HSO events generated by one chip are captured in the HSI unit of another. As long as separate, non-synchronized clock sources are used for each chip, the HSI line events will occur asynchronously to the chip.

To implement a test that could be either stand alone or co-resident without modification, the creation and verification of I/O events needed to be decoupled. Thus the basic structure of the **Dynamic Stability Test** takes the form of a set of I/O Producers causing events that I/O Consumers verify. Figure 4 gives a macro view of the Producer/Consumer relationship.

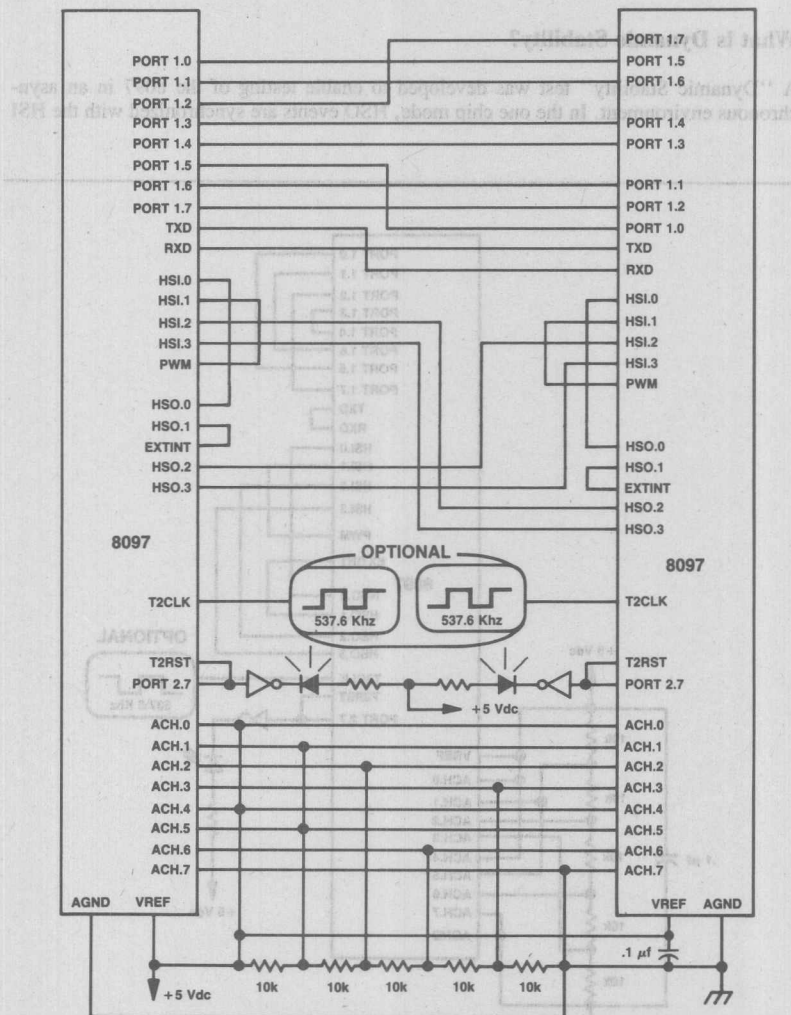


Figure 3. 8097 Strapback Configuration Dual Chip Mode

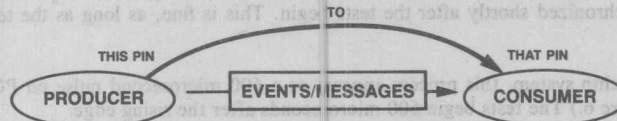


Figure 4. Producer/Consumer Relationship

What Does the Test Do?

Producer/Consumer exchanges were defined to test nearly all of the 8097 I/O capabilities concurrently. Following initialization, the transactions described are carried out by the set of interrupt service routines that make up the **Dynamic Stability Test**. The following section describes the test initialization. Then the tests performed are briefly described in the Producer/Consumer framework.

Initialization

To get the ball rolling, the background task must first CALL an initialization routine (DSTISR). This routine clears memory, executes the Selected Tests program (D96FST) from the **General Diagnostics**, and checks for the presence of an external clock on T2CLK. The serial port is then initialized for internal or external baud rate generation based on the presence of an external clock, and sign on messages are sent over the serial channel.

After initial tests are complete, and just prior to initiation of the interrupt service routines, a pulse is sent out on PORT1.3 that is used to synchronize controllers in the two chip mode. (See Figure 5.) Remember that the objective of the **Dynamic Stability Test** is to test the controllers asynchronously. Therefore, the synchronization is only done to insure that neither controller starts testing before both are ready to begin.

When a controller is ready to synchronize, it places a 0 on the PORT1.3 pin and looks for a 0 on its PORT1.4 pin. When a 0 is seen, the chip delays 600 microseconds, and then PORT1.3 is set high. The chip then loops until PORT1.4 also goes high. Another delay is inserted, and the tests begin. The worst skew between two controllers that can

7

SYNCHRONIZATION SEQUENCE IN THE DUAL CHIP MODE

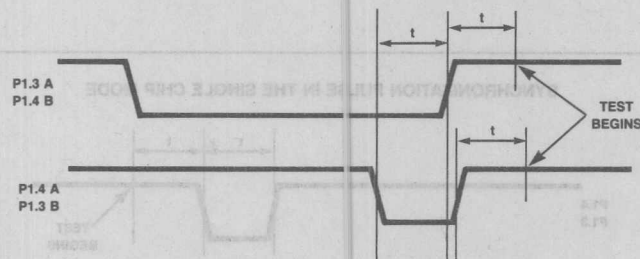


Figure 5. Dual Chip Synchronization

occur using this method is 9 state times ($2.25 \mu\text{s}$ in a 12 Mhz system). However, the skew should average between four and five state times. In any case, the parts will be far from synchronized shortly after the tests begin. This is fine, as long as the tests begin together.

In a one chip system, this process appears as a 600 microsecond pulse on PORT1.3. (See Figure 6.) The tests begin 600 microseconds after the rising edge.

When synchronization is complete, the interrupt service routines are initialized, interrupts are enabled, and control is returned to the background task. At this point, the testing really begins.

Producers and Consumers

The Producer/Consumer exchanges on the 8097 are executed by the interrupt service routines of the **Dynamimc Stability Test**. While some interrupt routines contain an entire Producer or Consumer, some are spread through many routines. Figure 7 shows on a broad level the transactions that occur during test execution. Short descriptions of each Producer and Consumer follow, along with an indication of which interrupt routines contain them.

Serial Producer •DSTSER• The Serial Producer constantly transmits a table of alphabetic and special characters, and test data which includes the current status of the test and the REAL TIME since reset.

Serial Consumer •DSTSER• The Serial Consumer monitors the data coming over the serial link to see if all the expected characters are transmitted correctly and in the correct order. Transmission of the test data and the REAL TIME is checked by counting characters between carriage returns.

Port1 Producer •DSTSWT• The Port1 Producer outputs a series of values on Port1 that are contained in a table constructed to test all possible combinations of input and output of ones and zeros. The test producer executes every 5000h TIMER1 counts via the expiration of Software Timer 1.

Port1 Consumer •DSTSWT• The Port1 Consumer verifies the patterns appearing on Port1 using a table which contains the expected values. The check executes every 1000h TIMER1 counts via the expiration of Software Timer 2.

A/D Producer •DSTSWT• The A/D Producer continually starts A/D conversions by loading an HSO command to initiate an A/D. The A/D Producer executes every time Software Timer 0 expires.

A/D Consumer •DSTA2D• The A/D Consumer verifies the result of conversions initiated by the A/D Producer. It then changes the channel set for conversion and loads an HSO command to cause a Software Timer 0 expiration.

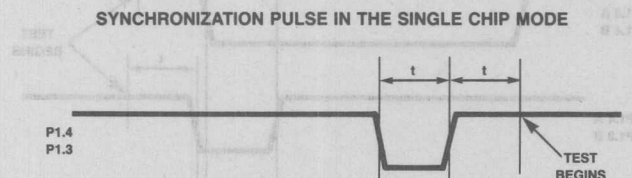


Figure 6. Single Chip Sync Pulse

External Interrupt Producer •DSTHSO• The External Interrupt Producer causes rising edges on HSO.1, which is tied to EXTINT. This Producer executes every time there has been a falling edge on HSO.1.

External Interrupt Consumer •DSTEXI• The External Interrupt Consumer responds to rising edges on EXTINT. It resets the WATCHDOG TIMER every execution and tests the Test Status Words every 30h executions to see that all tests are running. This Consumer also loads an HSO command to cause a falling edge on HSO.1.

PWM Producer •DSTTOV• The PWM Producer executes every time there is a timer overflow. In addition to changing the PWM period, it toggles an LED and checks for unexpected T2CLK overflows. There is no PWM Consumer per se, but the PWM output is tied to HSI.1 which is configured to clock T2CLK. In this way T2CLK counts at a known average rate, and is used by the test in a modulo count fashion to generate a real

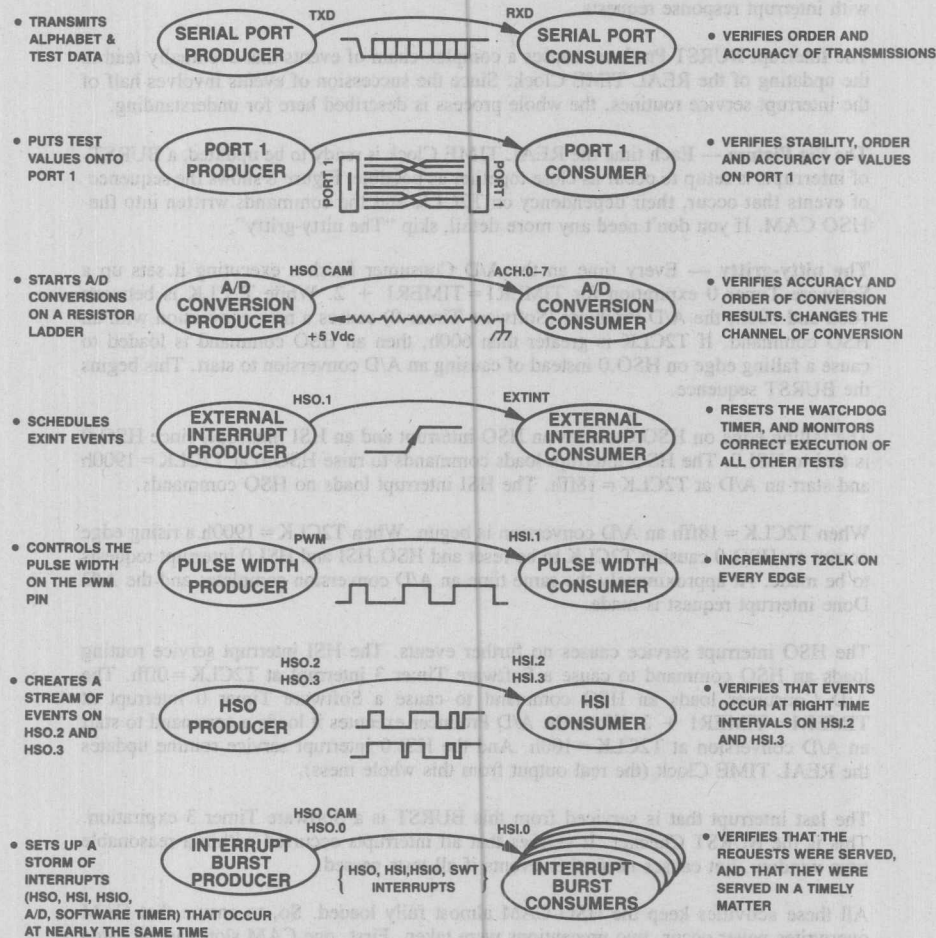


Figure 7. Producer/Consumer Overview

time clock. This module is also expandable to include tests that a user might want to execute only periodically.

HSO Producer •DSTHSO• The High Speed Output Producer executes every time an HSO event on HSO.2 or HSO.3 occurs. Varying pulse widths are created on the pins using predetermined tables of values. The minimum pulse width is 1000h; the maximum is 0C000h TIMER1 counts.

HSI Consumer •DSTHSI• The high speed inputs are monitored by the High Speed Input Consumer. The check executes every time an event occurs on HSI.2 or HSI.3. The HSI Consumer verifies that the proper pulse widths appear on the pins, and that the series of pulse widths is in the right order.

Interrupt BURST Producer •DSTSWT,DSTHI0,DSTHSO,DSTHSO• The previous Producer/Consumer transactions either go between controllers in the dual-chip mode, or stay within the same controller in the single-chip mode. However, there is one **Dynamic Stability Test** that executes invisibly to a co-controller in the dual-chip mode. This test, the Interrupt BURST Test, causes a flood of interrupts that almost fully load the 8097 with interrupt response requests.

The Interrupt BURST Producer causes a complex chain of events that eventually lead to the updating of the REAL TIME Clock. Since the succession of events involves half of the interrupt service routines, the whole process is described here for understanding.

The Big Picture — Each time the REAL TIME Clock is ready to be updated, a BURST of interrupts is setup to occur as close together as possible. Figure 8 shows the sequence of events that occur, their dependency on T2CLK and the commands written into the HSO CAM. If you don't need any more detail, skip "The nitty-gritty".

The nitty-gritty — Every time an the A/D Consumer finishes executing it sets up a Software Timer 0 expiration for $TIMER1 = TIMER1 + 2$. While T2CLK is between 100h and 600h, the A/D Producer (Software Timer 0) causes a new conversion with an HSO command. If T2CLK is greater than 600h, then an HSO command is loaded to cause a falling edge on HSO.0 instead of causing an A/D conversion to start. This begins the BURST sequence.

The falling edge on HSO.0 causes an HSO interrupt and an HSI interrupt, since HSO.0 is tied to HSI.0. The HSO interrupt loads commands to raise HSO.0 at $T2CLK = 1900h$ and start an A/D at $T2CLK = 18ffh$. The HSI interrupt loads no HSO commands.

When $T2CLK = 18ffh$ an A/D conversion is begun. When $T2CLK = 1900h$ a rising edge occurs on HSO.0 causing T2CLK to be reset and HSO, HSI and HSI.0 interrupt requests to be made. At approximately the same time an A/D conversion completes and the A/D Done interrupt request is made.

The HSO interrupt service causes no further events. The HSI interrupt service routing loads an HSO command to cause a Software Timer 3 interrupt at $T2CLK = 0ffh$. The A/D Consumer loads an HSO command to cause a Software Timer 0 interrupt at $TIMER1 = TIMER1 + 2$. When the A/D Producer executes it loads a command to start an A/D conversion at $T2CLK = 100h$. And the HSI.0 interrupt service routine updates the REAL TIME Clock (the real output from this whole mess).

The last interrupt that is serviced from this BURST is a Software Timer 3 expiration. This is the BURST Checker. It verifies that all interrupts occurred within a reasonable time window, but causes no further events if all tests passed.

All these activities keep the HSO CAM almost fully loaded. So, to ensure that CAM overwrites never occur, two precautions were taken. First, one CAM slot was allocated to four of the tests that use the HSO unit, and two slots were allocated for shared use by the Interrupt BURST process and the A/D conversion process.

The second precaution was to confirm that either the CAM was not full or the HOLDING REGISTER was empty (depending upon the test) before allowing any write to the CAM.

Figure 9 shows the HSO CAM loading over time, with T2CLK as the timebase. External Interrupt, Port1, HSO.2 and HSO.3 events each are allocated the use of one CAM slot all the time. While T2CLK is below 600h, but above 100h, another CAM slot is used by the A/D Done — Start A/D sequence. When T2CLK goes above 600h, two slots are used by the Interrupt BURST process. The BURST events conclude when T2CLK is reset and climbs to 100h. At 100h, the A/D Done — Start A/D sequence being again.

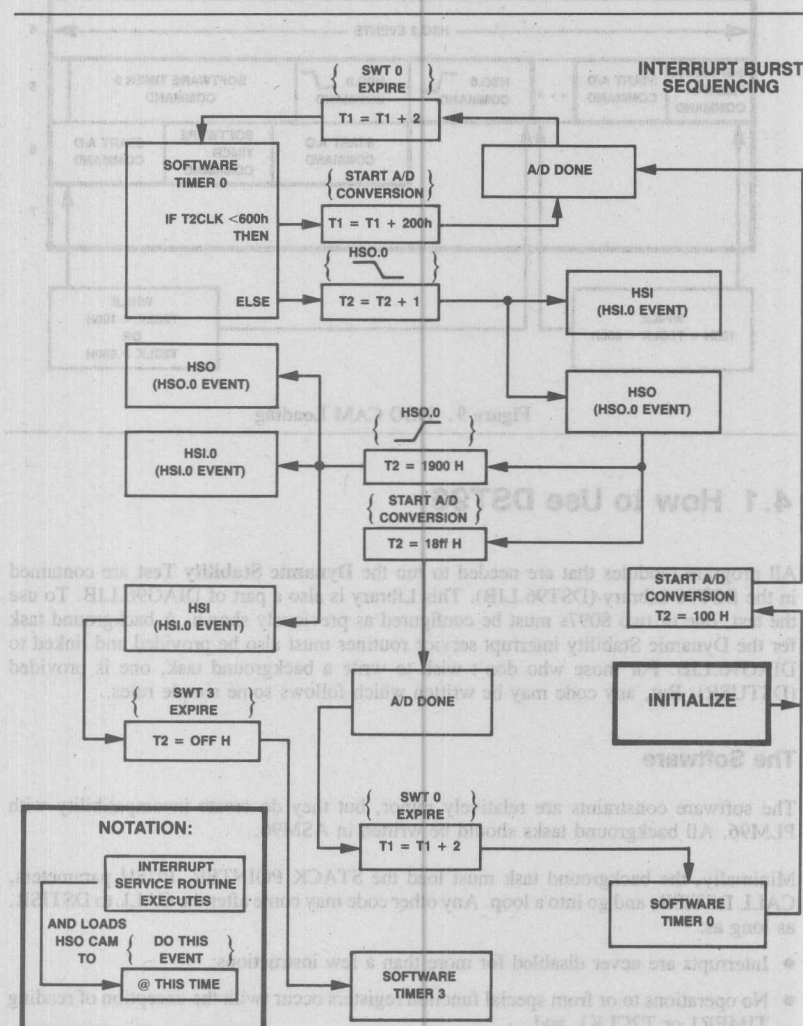


Figure 8. Interrupt BURST Sequencing

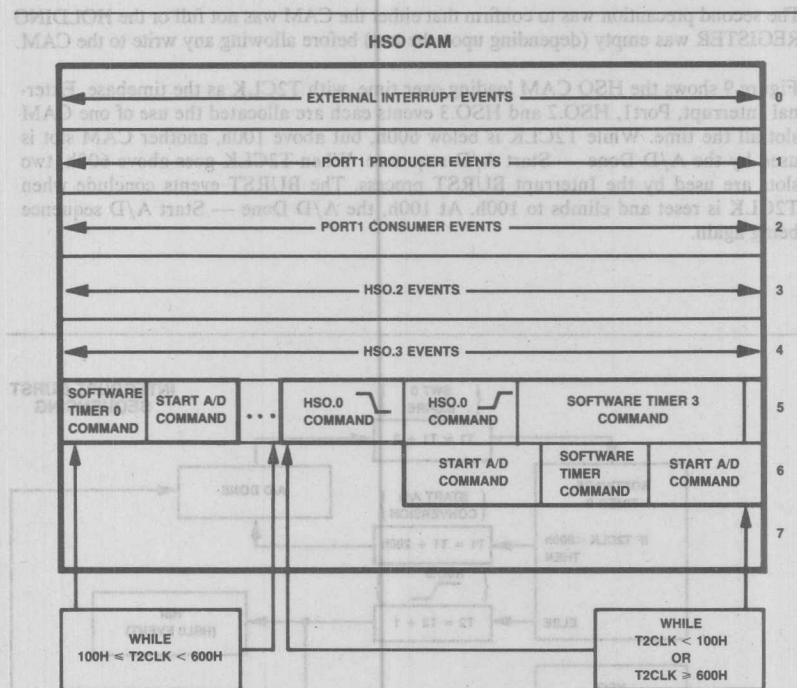


Figure 9. HSO CAM Loading

4.1 How to Use DST96

All program modules that are needed to run the **Dynamic Stability Test** are contained in the DST96 Library (DST96.LIB). This Library is also a part of DIAG96.LIB. To use the test, one or two 8097s must be configured as previously shown. A background task for the Dynamic Stability interrupt service routines must also be provided and linked to DIAG96.LIB. For those who don't wish to write a background task, one is provided (DSTUSR). But, any code may be written which follows some simple rules.

The Software

The software constraints are relatively minor, but they do create incompatibility with PLM96. All background tasks should be written in ASM96.

Minimally, the background task must load the STACK POINTER, PUSH parameters, CALL DSTISR, and go into a loop. Any other code may come after the CALL to DSTISR, as long as:

- Interrupts are never disabled for more than a few instructions;
- No operations to or from special function registers occur (with the exception of reading TIMER1 or T2CLK), and

Other less grave limitations on the main task are that it:

- Be CSEged at 2080h;

- Write only to EREG1, EREG2, OSEG registers from 40h to 5Ch, or external RAM, (the OSEG is an RL96 technicality, once DSTISR returns control to the MAIN TASK, locations 40h to 5Ch are not touched by the Tests); other registers can be read, but not written to;
- Communicate to the outside world through Port3 and Port4, (these Ports are untouched by the tests), or memory mapped I/O registers;

To provide the **Dynamic Stability Test** modules for linkage to your program, modify the batch file DSTRL.BAT to suit your system with respect to memory mapping and invoke the batch file with the appropriate background task filename. For example, type:

```
DSTRL DSTUSR
```

The Hardware

The **Dynamic Stability Test** has been designed to allow flexibility in the way output from the tests is used.

Minimally, no output device (printer, terminal) or function generators need to be attached to the test. If the LED attached to Port 2.7 is not flashing, the test failed. However, no other information may be gained.

To support a greater level of debugging (of the test code initially), the test was designed to output status and error information to one 4800 and one 300 baud device. The baud rates are derived from the function generators if present. Figure 10 shows how both devices can be attached to the test.

With this configuration, the test outputs an initialization message to both devices, then selects just the 4800 baud line for monitoring the Serial Port Producer/Consumer transactions. If an error is detected, the 300 baud line is selected for an error information dump.

A diagram of the circuit used in developing the **Dynamic Stability Test** appears in Figure 11. It is sufficiently general purpose for use in either the single or double chip modes, with or without printers or terminals attached.

The circuit requires that the 8097 I/O signals be present on an SBE-96 compatible 50 pin connector. The circuit also assumes that the analog voltage reference is provided through the cable. Therefore, if you are using the SBE-96, the jumpers to do this need to be in place (jumper numbers vary with the SBE-96 version).

Figure 12 describes how to jumper the **Dynamic Stability Test** board for one or two chip tests. Figure 13 shows the SBE-96 50 pin connector pinout. The following sections describe in detail the actions of each interrupt service routine in implementing the Producer/Consumer transactions.

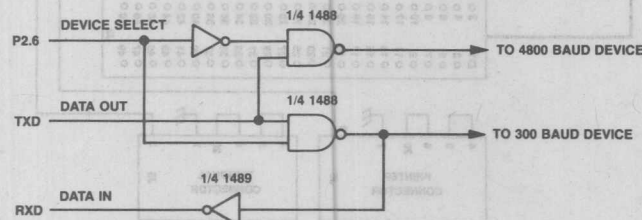
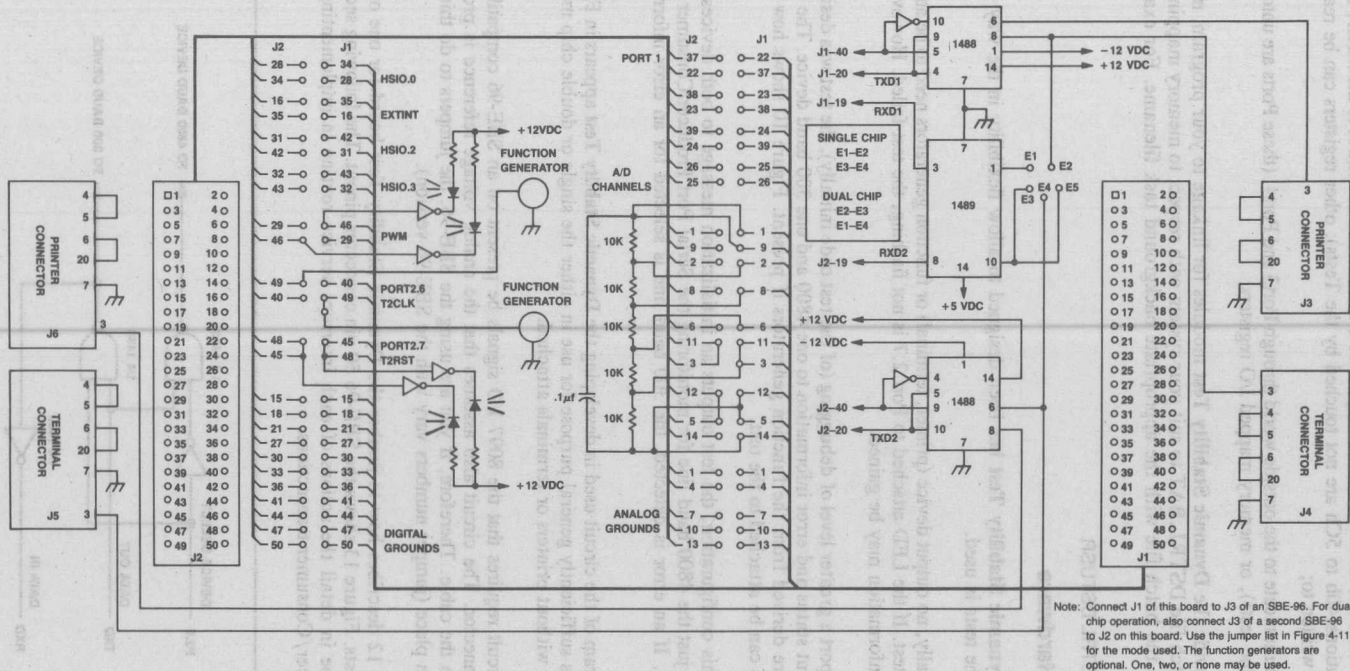


Figure 10. Output Device Selection Circuit



Jumper Connections for Single Chip Mode

J1	
22 - 37	34 - 28
23 - 38	35 - 16
24 - 39	42 - 31
25 - 26	43 - 32
45 - 48	46 - 29
1 - 4 - 7 - 10 - 13	
15 - 18 - 21 - 27 - 30 - 33 - 36 - 41 - 44 - 47 - 50	

Also

E1 - E2

E3 - E4

Jumper Connections for Dual Chip Mode

J1 - J2	J1 - J2	J1 - J2	J1	J2 - J1
22 - 37	33 - 33	14 - 14	34 - 28	22 - 37
23 - 38	36 - 36	4 - 4	45 - 48	23 - 38
24 - 39	41 - 41	5 - 5	46 - 29	24 - 39
25 - 26	44 - 44	7 - 7	35 - 16	25 - 26
42 - 31	47 - 47	10 - 10	J2	42 - 31
43 - 32	50 - 50	13 - 13		43 - 32
15 - 15	1 - 1		34 - 28	
18 - 18	9 - 9		45 - 48	
21 - 21	2 - 2		46 - 29	
27 - 27	8 - 8		35 - 16	
30 - 30	6 - 6		Also	
	11 - 11			
	3 - 3		E2 - E5	
	12 - 12		E1 - E4	

Figure 12. Dynamic Stability Board Jumper List

ANALOG GROUND	1	20	ANALOG CHANNEL 3
ANALOG CHANNEL 1	03	40	ANALOG GROUND
ANALOG CHANNEL 0	05	60	ANALOG CHANNEL 2
ANALOG GROUND	07	80	ANALOG CHANNEL 6
ANALOG CHANNEL 7	09	100	ANALOG GROUND
ANALOG CHANNEL 5	11	120	ANALOG CHANNEL 4
ANALOG GROUND	13	140	ANALOG VREF
DIGITAL GROUND	15	160	EXTERNAL INTERRUPT
RESET	17	180	DIGITAL GROUND
RXD	19	200	TXD
DIGITAL GROUND	21	220	PORT 1.0
PORT 1.1	23	240	PORT 1.2
PORT 1.3	25	260	PORT 1.4
DIGITAL GROUND	27	280	HSI.0
HSI.1	29	300	DIGITAL GROUND
HSO.4/HSI.2	31	320	HSO.5/HSI.3
DIGITAL GROUND	33	340	HSO.0
HSO.1	35	360	DIGITAL GROUND
PORT 1.5	37	380	PORT 1.6
PORT 1.7	39	400	PORT 2.6
DIGITAL GROUND	41	420	HSO.2
HSO.3	43	440	DIGITAL GROUND
PORT 2.7	45	460	PWM/PORT 2.5
DIGITAL GROUND	47	480	T2RST
T2CLK	49	500	DIGITAL GROUND

Figure 13. SBE-96 J3 Pinout

DST Initialization (DSTISR)

Brief Description:

This module is the invocation and initialization code for the Dynamic Stability Test.

Assembly Language Calling Sequence:

```
PUSH    <RAM segment1 starting address>
PUSH    <RAM segment1 ending address>
PUSH    <RAM segment2 starting address>
PUSH    <RAM segment2 ending address>
PUSH    <random seed>
PUSH    <random test length>
PUSH    <top of code address>
PUSH    <argument1 for Multiply/Divide Core test>
PUSH    <argument2 for Multiply/Divide Core test>
PUSH    <bit pattern for memory test>
CALL    DSTISR
```

When All Tests Pass:

```
EREG1 := 0040h
EREG2 := 0000h
```

When a Test Fails:

```
EREG1 := 0140h on abnormal RESET      EREG2 := TIMER1
EREG1 := 0240h if T2CLK won't change  EREG2 := xxxxh
EREG1 := 0340h if T2RST did not work   EREG2 := xxxxh
EREG1 := 0440h if IOC0.1 did not work  EREG2 := xxxxh
```

Detailed Description:

This module initializes the registers used by **Dynamic Stability Test** Modules, checks to see if there is an external clock present, tests T2CLK counting and reset functionality, and outputs initialization messages to the two output devices. The selected tests module (D96FST) from the **General Diagnostics** is also executed using the parameters specified.

When all initialization tests are passed, then a synchronization is performed to place the two processors in a dual-chip mode test in close sync. The PORT1 pins are used as to perform the handshaking synchronization. After synchronization, all **Dynamic Stability Tests** are activated and control is returned to the user program.

External Interrupts (DSTEXI)

Brief Description:

This module executes every time there is a rising edge on the EXTINT pin. The test resets the WATCHDOG TIMER and verifies execution of all Dynamic Stability routines.

If Test Fails:

EREG1 := 01A0h if a test did not execute

EREG2 := Number of Shifts done

Detailed Description:

This routine executes every time there is a rising edge on the EXTINT pin, causing an external interrupt. Each execution, the WATCHDOG TIMER is reset and an HSO command to clear the HSO.1 pin in 1000h TIMER1 counts is loaded into the CAM. The HSO routine that responds to that event will cause HSO.1 to go high, thus causing another vector to DSTEXI.

Every 30h executions of this module, the Test Status Words are NOTed and then NORMaLized to see if any test did not execute. If any bit in the Test Status Words is left set after being complemented, the NORML instruction will leave the most significant bit set, indicating an error. If there was no error, the TSWORDs are cleared. The user can change a mask in DSTEXI to enable checking of any of the currently spare bits in TSWORD. The TSWORD bit map is as follows:

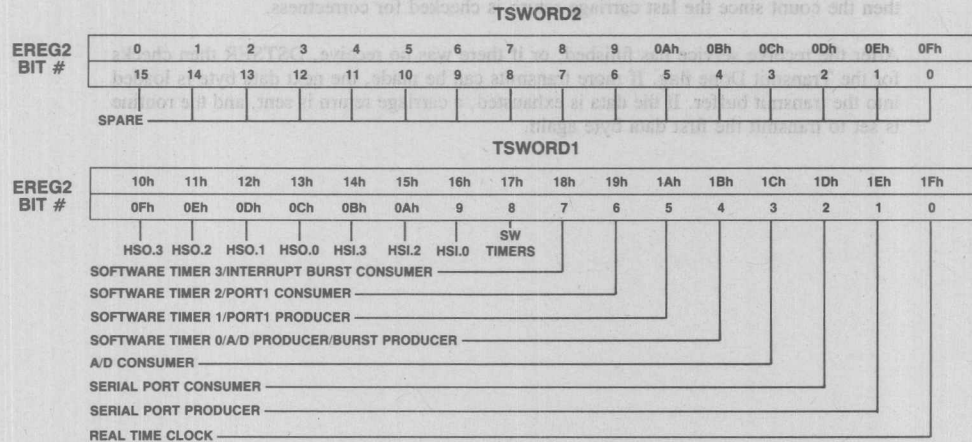


Figure 14. Test Status Word Bit Map

Serial Port (DSTSER)**Brief Description:**

This module contains the Serial Port Consumer and Producer routines for the **Dynamic Stability Tests**. It is executed on every Serial Interrupt.

If Test Fails:

EREG1 := 01B0h if a bad character was received

EREG2 := actual received character

EREG1 := 02B0h if an incorrect number of characters
came between carriage returns

EREG2 := actual count

Detailed Description:

This interrupt service routine executes every time there is a Serial Interrupt. The data that is transmitted and checked by the test consists of first, the alphabet and some special characters; second, the current REAL TIME; and finally, the bit representation of the Test Status Words. The receiver verifies the alphabet and funny characters and counts characters until a carriage return. The following is an example of what the output looks like.

ABCDEF GHIJKLMNOPQRSTUVWXYZ*%&[]@001:23:59.61 1111110111011111110001111

The code first checks for a Receive Done flag. If a receive just completed, the receive buffer is emptied and checked for validity. If the received character is a carriage return, then the count since the last carriage return is checked for correctness.

After the receive service has finished, or if there was no receive, DSTSER then checks for the Transmit Done flag. If more transmits can be made, the next data byte is loaded into the transmit buffer. If the data is exhausted, a carriage return is sent, and the routine is set to transmit the first data byte again.

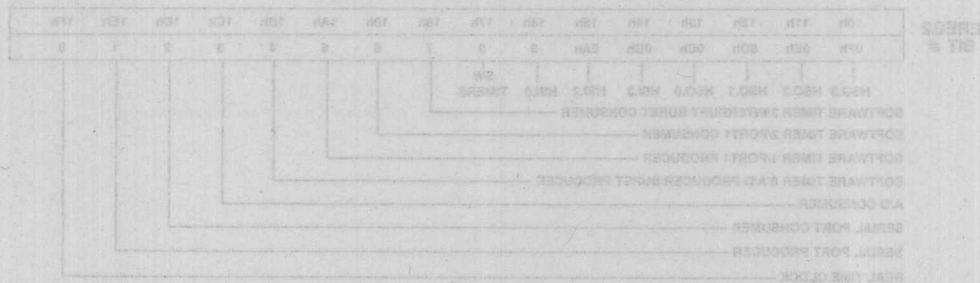


Figure 14. Test Status Word Bit Map

Software Timers (DSTSWT)**Brief Description:**

This module is executed every time a Software Timer Interrupt expires. The routine includes the Port1 Producer and Consumer, the A/D producer, and the Interrupt Burst control and verification code.

If a Test Fails:

EREG1 := 01D0h If an unexpected value is found on Port 1
EREG2 := expected value in high byte, actual value in low byte
EREG1 := 02D0h A/D Done interrupt did not occur within BURST window
EREG2 := Time between A/D done and Software Timer 0
EREG1 := 03D0h REAL TIME update did not occur within BURST window
EREG2 := Time between REAL TIME update and Software Timer 0
EREG1 := 04D0h HSO.0 response did not occur within BURST window
EREG2 := Time between HSO.0 interrupt and Software Timer 0
EREG1 := 05D0h HSI(.0) response did not occur within BURST window
EREG2 := Time between HSI(.0) service and Software Timer 0
EREG1 := 01D1h Invalid T2CLK value reached
EREG2 := T2CLK found
EREG1 := 02D1h Test reached an illegal Software Timer 0 state
EREG2 := the illegal case jump that was made

Detailed Description:

This module is called every time a Software Timer expires and causes an interrupt. Software timers are used by the A/D Done — A/D Trigger Sequence, the Interrupt Burst Sequence, and the Port1 Producer and Port1 Consumer.

When Software Timer 0 expires, a case jump is done on the BURST_STATE variable to sequence the A/D and interrupt BURST process to the appropriate state. Depending upon the value of T2CLK and the state of the A/D converter, either an A/D conversion is initiated or HSO.0 is set to go low to begin the interrupt BURST events.

When Software Timer1 expires, a new value is written to Port1 from a table constructed to test all combinations of input/output states on the quasi-bidirectional port pins. The HSO CAM is also loaded with a command to cause Software Timer1 to overflow again in 5000h TIMER1 counts.

When Software Timer 2 expires, Port1 is read and compared to a table of expected entries. If the value is correct, then an HSO command is loaded into the CAM to cause another Software Timer 2 expiration in 1000h TIMER1 counts. If the value is not correct, the next entry in the Table is checked. If there is still no match, an error is reported. If there is a match, the CAM loading occurs and Software Timer 3 is checked for expiration.

If Software Timer 3 has expired, then the flurry of BURST interrupts should have just occurred. The routine checks to see that each event happened within a reasonable time window. If the checks pass, then the routine exists with no further action.

Real Time Clock (HSI0) (DSTHI0)**Brief Description:**

This routine executes every time there is a rising edge on HSI.0 and updates the real time clock value.

When Module Executes:

REAL_TIME := **REAL_TIME** + .204 seconds

Detailed Description:

This module is the HSI.0 interrupt service routine. On each rising edge of HSI.0, the value in the REAL TIME clock buffer is updated to reflect the passing of 1900h T2CLK counts. Since the PWM output is tied to T2CLK, and the average time between edges is 31.875 μ s in a 12 MHz system, then 1900h T2CLK counts represents .204 seconds.

Execution of this module occurs during the interrupt BURST events. No action other than updating the REAL TIME clock is taken in this routine.

High Speed Outputs (DSTHSO)**Brief Description:**

This module manages the pulse width outputs on HSO.2 and HSO.3, and causes the Manager test to execute.

Detailed Description:

Every time an HSO command is executed that has the Interrupt bit set, this program executes. The routine manages the pulse widths on HSO lines two and three, and causes the Manager module to execute at the right time.

When a falling edge has been caused on either HSO.2 or HSO.3, DSTHSO loads a command into the CAM to cause a rising edge on the same line at a time that gives the line a low pulse width equal to a predetermined table value. Rising edges cause analogous responses. The tables used cause low and high pulse widths that vary from 1000h and 0C000h. The length of the tables differ by one so that all combinations of low and high table times occur.

When a falling edge was caused on HSO.1, the routine loads a command into the CAM to cause a rising edge on the same line two TIMER1 counts later. Since HSO.1 is tied to the EXTINT pin, rising edges cause the Manager Routine to execute.

High Speed Inputs (DSTHSI)**Brief Description:**

This module does the verification of events on the HSI lines and initiates some interrupt BURST events when appropriate.

If a Test Fails:

EREG1 := 0161h if a high pulse on HSI.2 had an unexpected width
EREG2 := difference between actual and expected pulse width

EREG1 := 0261h if a low pulse on HSI.2 had an unexpected width
EREG2 := difference between actual and expected pulse width

EREG1 := 0361h if a high pulse on HSI.3 had an unexpected width
EREG2 := difference between actual and expected pulse width

EREG1 := 0461h if a low pulse on HSI.3 had an unexpected width
EREG2 := difference between actual and expected pulse width

EREG1 := 0561h if the HSI unit indicated that an HSI.1 event occurred
EREG2 := the time recorded in the FIFO

Detailed Description:

This module executes every time an event is loaded into the HSI Holding Register. Verification of pulse widths on HSI.2 and HSI.3 is done from tables of expected values. Any deviation is reported as an error.

If the test detects a negative transition on HSI.0, then commands are loaded into the HSO CAM to start an A/D at T2CLK = 18ffh and to set HSO.0 high at T2CLK = 1900h. This results in an HSO, HSI, HSI.0 and A/D Done interrupt requests to occur at approximately the same time — approaching a full demand on interrupt service.

When a rising edge on HSI.0 is detected, an HSO command is loaded into the CAM to cause a Software Timer 3 interrupt when T2CLK = 100h. The Software Timer 3 interrupt service will check to see that all burst events happened fast enough.

HSI.1 events are disabled from the FIFO. Any event detected on this line is reported as an error.

A/D Conversion Complete (DSTA2D)**Brief Description:**

This module executes every time an A/D conversion is complete. The conversion result is checked for correctness, the A/D converter is setup to convert on the next channel when initiated by an HSO command, and an HSO command to cause a Software Timer 0 expiration is loaded.

If Test Fails:

EREG1 := 01C0h on a conversion error

EREG2 := channel on which error occurred

Detailed Description:

This module executes every time an A/D conversion is complete. The conversion result is checked against a test table for correctness, and the A/D converter is setup to convert on the next channel when initiated by an HSO command. An HSO command to cause a Software Timer 0 expiration in 0002h TIMER1 counts is loaded just prior to exiting the module.

While T2CLK has a value between 100h and 600h, A/D conversions are initiated by the Software Timer 0 Interrupt service routine. When T2CLK goes above 600h, an A/D conversion is initiated by the HSO.0 interrupt service routine.

Given the possibility of additive error in 5% resistors, the conversion is tested to only six bits of accuracy.

Timer Overflows (DSTTOV)**Brief Description:**

This module toggles a port pin tied to an LED, manages the PWM output, performs some simple tests, and is expandable to allow inclusion of user written tests.

If Test Fails:

EREG1 := 0190h if T2CLK had an overflow indication

EREG2 := T2CLK a the time the error was found

Detailed Description:

This module executes every time TIMER1 or T2CLK overflow. Only TIMER1 overflows are valid however, so T2CLK overflows are flagged as an error. Each overflow, a new period is loaded into the PWMCONTROL register from a table of pulse periods. If an LED is connected, it will appear to slowly change in intensity. Port2.7 is also toggled in this routine to light another LED.

This interrupt routine can be expanded with special tests that are to execute on a periodic basis. Any of the spare bits in the Test Status Words can also be used by specialized tests. They will be checked by the External Interrupt service routine with a simple change in a bit mask.

Macro Module (DSTMAC)**Brief Description:**

This module contains four macros used by the **Dynamic Stability Test**.

Assembly Language Invocation:

SPSTATUS	Temp_Register
or	
SPWAIT	(RI, TI)
or	
BR_ON_ERROR	Label
or	
RESET_WATCHDOG	

Detailed Description:

The SPSTATUS Macro is used to ORB the Serial Port Status Register to a temp register. The Macro needs to be used to work around a bug in the 809x-90.

The SPWAIT Macro is used to cause program execution to halt and wait for an RI or TI flag, depending upon which is specified.

The BR_ON_ERROR Macro tests the high byte of EREG1 and jumps to the label if the byte is not zero. This can be used every time a **General Diagnostic** completes since the detection of any error will cause the high byte of EREG1 to be non-zero.

The RESET_WATCHDOG Macro does just what it says. The WATCHDOG TIMER is reset by writing the correct sequence to location 0Ah.

To access a DSTMAC macro, this module must be \$INCLUDED.

Brief Description:

This module is called if any error is detected in the **Dynamic Stability Test**. Information about the error is output over the serial port, and the test is restarted.

Assembly Language Calling Sequence:

CALL Error_Proc

Detailed Description:

This module is CALLED on detection of any error in the **Dynamic Stability Test**. When CALLED, the procedure:

- disables interrupts,
- saves any rapidly changing values (TIMER1,T2CLK,HSO_STATUS, . . .),
- waits for a serial transmit in progress to complete,
- waits for the current serial receive to complete,
- empties eight entries from the HSI_FIFO,
- transmits an open loop sync sequence in case a co-controller is stuck in the sync routine, and
- waits a few hundred milliseconds to ensure that a co-controller has also detected a failure.

After these steps have been taken, the DSTERR de-selects the 4800 baud line, selects the 300 baud line, and outputs error messages. These messages include the Error Code (EREG1), the Detail Code (EREG2), the address of the line in the test which found the error, and the REAL TIME since reset.

Following the error messages, the procedure dumps the data contained in the registers and the external error buffer out over the serial port to the 300 baud device.

Finally, a RST instruction followed by a branch to the RST instruction is executed. If the WATCHDOG TIMER is externally disabled, the test will stay in this loop. If the WATCHDOG TIMER is not disabled, the test chip will reset, and the **Dynamic Stability Test** will reinitialize.

DST Example User Code (DSTUSR)

Brief Description:

This is an example program that initiates the **Dynamic Stability Test** and then executes some **General Diagnostics** as a background task.

Detailed Description:

DSTUSR sends parameters defined at assembly time to the DST initialization routine (DSTISR). When control returns to DSTUSR, the example repeatedly executes ALU01, ALU02, ALU04, ALU05 and MEM0A. It takes two minutes (with the given memory parameters) for the DSTUSR background task to cycle once while interrupts are running.

When creating a custom background task, using this example program as a template will speed development.

APPENDICES

APPENDIX A • DIAG96.LIB Error Messages by EREG1 Code

APPENDIX B • DIAG96.LIB Error Messages by Module Name

APPENDIX C • Description of DIAG96.LIB Batch Files

APPENDIX D • Example Program Listings

— D96A96
— D96P96
— D96FST
— DSTUSR

APPENDIX A

DIAG96.LIB Error Messages by EREG1 Code

0000	No Message EREG2 = 0ffffh MODULE = SYS01/Common Symbols
0002	All Tests Passed EREG2 = 0000 MODULE = SYS02/System Power-up
0003	All Tests Passed EREG2 = 0000 MODULE = SYS03/Program Counter
0011	All Tests Passed EREG2 = 0000 MODULE = ALU01/Add/Subtract
0012	All Tests Passed EREG2 = 0000 MODULE = ALU02/MULUB
0013	All Tests Passed EREG2 = 0000 MODULE = ALU03/Multiply/Divide Table
0014	All Tests Passed EREG2 = 0000 MODULE = ALU04/Multiply/Divide Random
0015	All Tests Passed EREG2 = 0000 MODULE = ALU05/Multiply/Divide Core
0021	All Tests Passed EREG2 = 0000 MODULE = MEM01/Complementary Address (Registers)
0022	All Tests Passed EREG2 = 0000 MODULE = MEM02/Walking Ones/Zeros (Registers)
0023	All Tests Passed EREG2 = 0000 MODULE = MEM03/Galloping Ones/Zeros (Registers)
0024	No bits were set in the byte tested EREG2 = 0000 MODULE = MEM04/Bits Set
0025	All Tests Passed EREG2 = 0000 MODULE = MEM05/Checkerboard Pattern (Registers)
0026	All Tests Passed EREG2 = 0000 MODULE = MEM06/Complementary Address

MCS®-96 Diagnostics Library

0027	All Tests Passed EREG2=0000 MODULE=MEM07/Walking Ones	A signed operation failed EREG2=offending argument on error MODULE=ALU03/Multiply/Divide Table	0110
0028	All Tests Passed EREG2=0000 MODULE=MEM08/Galloping Ones	A signed operation failed EREG2=offending argument on error MODULE=ALU03/Multiply/Divide Table	0110
0029	All Tests Passed EREG2=0000 MODULE=MEM09/Walking Ones/Zeros	A memory location failed EREG2=address of the error MODULE=MEM09/Walking Ones/Zeros	0110
002A	All Tests Passed EREG2=0000 MODULE=MEM0A/Galloping Ones/Zeros	A memory location failed EREG2=address of the error MODULE=MEM0A/Galloping Ones/Zeros	0110
002C	All Tests Passed EREG2=0000 MODULE=MEM0C/User Pattern (Registers)	A memory location failed EREG2=address of the error MODULE=MEM0C/User Pattern (Registers)	0110
002D	All Tests Passed EREG2=0000 MODULE=MEM0D/User Pattern	A memory location failed EREG2=address of the error MODULE=MEM0D/User Pattern	0110
0030	All Tests Passed, checksum is ready EREG2=16-bit checksum MODULE=D96A96/ALL Tests in ASM96	A memory location failed EREG2=address of the error MODULE=D96A96/ALL Tests in ASM96	0110
0040	Initialization completed satisfactorily EREG2=0000 MODULE=DSTISR/DST Initialization	A memory location failed EREG2=address of the error MODULE=DSTISR/DST Initialization	0110
00E0	All Tests Passed, checksum is over range specified EREG=16-bit checksum MODULE=D96FST/Selected Tests in ASM	A memory location failed EREG2=address of the error MODULE=D96FST/Selected Tests in ASM	0110
00F0	All Tests Passed, checksum is ready EREG2=16-bit checksum MODULE=D96P96/ALL Tests in PLM96	A memory location failed EREG2=address of the error MODULE=D96P96/ALL Tests in PLM96	0110
0102	I/O Status Registers were unexpected EREG2=IOS0 in low byte, IOS1 in high byte MODULE=SYS02/System Power-up	A memory location failed EREG2=address of the error MODULE=SYS02/System Power-up	0110
0103	Test Code Returned Early EREG2=Early Time MODULE=SYS03/Program Counter	A memory location failed EREG2=address of the error MODULE=SYS03/Program Counter	0110
0111	An Addition error occurred EREG2=offending argument when the error occurred MODULE=ALU01/Add/Subtract	16-bit Checksum is ready EREG2=offending argument when the error occurred MODULE=ALU01/Add/Subtract	0110
0112	Incorrect multiplication result was detected EREG2=Multiplier/Multiplicand MODULE=ALU02/MULUB	A memory location failed EREG2=address of the error MODULE=ALU02/MULUB	0110
0115	A signed operation failed EREG2=offending argument on error MODULE=ALU03/Multiply/Divide Table	A memory location failed EREG2=address of the error MODULE=ALU03/Multiply/Divide Table	0110

MCS®-96 Diagnostics Library

0115	A signed operation failed EREG2 = offending argument on error MODULE = ALU04/Multiply/Divide Random	All Tests Passed EREG2 = 0000 MODULE = MEM07	0027
0115	A signed operation failed EREG2 = offending argument on error MODULE = ALU05/Multiply/Divide Core	All Tests Passed EREG2 = 0000 MODULE = MEM08	0028
0121	A memory location failed EREG2 = address of the error MODULE = MEM01/Complementary Address (Registers)	All Tests Passed EREG2 = 0000 MODULE = MEM01	0029
0122	A memory location failed EREG2 = address of the error MODULE = MEM02/Walking Ones/Zeros (Registers)	All Tests Passed EREG2 = 0000 MODULE = MEM02	002A
0123	A memory location failed EREG2 = address of the error MODULE = MEM03/Galloping Ones/Zeros (Registers)	All Tests Passed EREG2 = 0000 MODULE = MEM03	002C
0124	At least one bit was set in the byte tested EREG2 = number of bits set MODULE = MEM04/Bits Set	All Tests Passed EREG2 = 0000 MODULE = MEM04	002D
0125	A memory location failed EREG2 = address of the error MODULE = MEM05/Checkerboard Pattern (Registers)	All Tests Passed, checksum is ready EREG2 = 16-bit checksum MODULE = MEM05	0030
0126	A memory location failed EREG2 = address of error MODULE = MEM06/Complementary Address	Initialization completed satisfactorily EREG2 = 0000 MODULE = MEM06	0040
0127	A memory location failed EREG2 = address of the error MODULE = MEM07/Walking Ones	All Tests Passed, checksum is over EREG2 = 16-bit checksum MODULE = MEM07	00E0
0128	A memory location failed EREG2 = address of the error MODULE = MEM08/Galloping Ones	All Tests Passed, checksum is ready EREG2 = 16-bit checksum MODULE = MEM08	00F0
0129	A memory location failed EREG2 = address of the error MODULE = MEM09/Walking Ones/Zero	NO Status Registers were unexpected EREG2 = 1080 in low byte, 1081 MODULE = MEM09	0102
012A	A memory location failed EREG2 = address of the error MODULE = MEM0A/Galloping Ones/Zeros	Test Code Returned Early EREG2 = Early Time MODULE = MEM0A	0103
012B	16-bit Checksum is ready EREG2 = 16-bit Checksum MODULE = MEM0B/Checksum	An Addition error occurred EREG2 = offending argument when MODULE = ALU01/Add/Subtract	0110
012C	A memory location failed EREG2 = address of the error MODULE = MEM0C/User Pattern (Registers)	Incorrect multiplication result was detected EREG2 = Multiplier/Multiplicand MODULE = ALU03	0112
012D	A memory location failed EREG2 = address of the error MODULE = MEM0D/User Pattern	A signed operation failed EREG2 = offending argument on MODULE = ALU03/Multiply/Divide	0113

0140	An abnormal RESET occurred EREG2=TIMER1 MODULE=DSTISR/DST Initialization	1830
0161	A high pulse on HSI.2 had an unexpected width EREG2= difference between actual and expected pulse width MODULE=DSTHSI/High Speed Inputs	0280
0190	An overflow of T2CLK was indicated EREG2=TIMER1 MODULE=DSTTOV/Timer Overflows	0300
01A0	One or more DST Module did not execute on time EREG2= Number of SHIFTs done MODULE=DSTEXI/External Interrupt (Supervisor)	0301
01B0	An unexpected serial character was received EREG2= Bad character received MODULE=DSTSER/Serial Port	0302
01C0	An unexpected A/D conversion result was found EREG2= Channel number of unexpected result MODULE=DSTA2D/A/D Conversion Complete	0303
01D0	Found unexpected value on PORT1 EREG2= expected value in high byte, actual in low byte MODULE=DSTSWT/Software Timers	0311
01D1	Invalid T2CLK value reached EREG2= T2CLK MODULE=DSTSWT/Software Timers	0312
0202	TIMER1 did not change over time EREG2= TIMER1 MODULE=SYS02/System Power-up	0313
0203	Test Code Returned Late EREG2= Late Time MODULE=SYS03/Program Counter	0315
0211	A Subtraction error occurred EREG2= offending argument when the error occurred MODULE=ALU01/Add/Subtract	0340
0215	An unsigned operation failed EREG2= offending argument on error MODULE=ALU03/Multiply/Divide Table	0361
0215	An unsigned operation failed EREG2= offending argument on error MODULE=ALU04/Multiply/Divide Random	0361
0215	An unsigned operation failed EREG2= offending argument on error MODULE=ALU05/Multiply/Divide Core	0361
0240	T2CLK will not change EREG2= xxxx MODULE=DSTISR/DST Initialization	0403

MCS®-96 Diagnostics Library

0261	A low pulse on HSI.2 had an unexpected width EREG2 = difference between actual and expected pulse width MODULE = DSTHSI/High Speed Inputs	0261
02B0	A carriage return was received out of sequence EREG2 = number of characters since a carriage return MODULE = DSTSER/Serial Port	02B0
02D0	A/D Done did not occur within BURST window EREG2 = Time between A/D done and Software Timer 0 MODULE = DSTSWT/Software Timers	02D0
02D1	Test reached an illegal Software Timer 0 state EREG2 = Illegal case jump made MODULE = DSTSWT/Software Timers	02D1
0302	Zero Register was found to change EREG2 = Program Status Word At Failure MODULE = SYS02/System Power-up	0302
0303	Counter Register contained unexpected value EREG2 = Erroneous Counter Value MODULE = SYS03/Program Counter	0303
0311	A flag error occurred EREG2 = offending argument when the error occurred MODULE = ALU01/Add/Subtract	0311
0315	A flag error occurred EREG2 = offending argument on error MODULE = ALU03/Multiply/Divide Table	0315
0315	A flag error occurred EREG2 = offending argument on error MODULE = ALU04/Multiply/Divide Random	0315
0315	A flag error occurred EREG2 = offending argument on error MODULE = ALU05/Multiply/Divide Core	0315
0340	T2RST pin would not RESET T2CLK EREG2 = xxxx MODULE = DSTISR/DST Initialization	0340
0361	A high pulse on HSI.3 had an unexpected width EREG2 = difference between actual and expected pulse width MODULE = DSTHSI/High Speed Inputs	0361
0391	Illegal Opcode	0391
03D0	REAL TIME update did not occur within BURST window EREG2 = Time between REAL TIME update and Software Timer 0 MODULE = DSTSWT/Software Timers	03D0
0402	PUSHF or POPF failed EREG2 = Erroneous PUSHed or POPed value found MODULE = SYS02/System Power-up	0402

MCS[®]-96 Diagnostics Library

0440	I0C0.1 would not RESET T2CLK EREG2 = xxxx MODULE = DSTISR/DST Initialization
0461	A low pulse on HSI.3 had an unexpected width EREG2 = difference between actual and expected pulse width MODULE = DSTHSI/High Speed Inputs
04D0	HSO.0 response did not occur within BURST window EREG2 = Time between HSO.0 update and Software Timer 0 MODULE = DSTSWT/Software Timers
0502	Sticky Bit would not set EREG2 = 3fffh MODULE = SYS02/System Power-up
0502	Sticky Bit would not clear EREG2 = 0000 MODULE = SYS02/System Power-up
0561	HSI unit indicated an HSI.1 event occurred EREG2 = Time recorded in HSI FIFO MODULE = DSTHSI/High Speed Inputs
05D0	HSI(.0) response did not occur within BURST window EREG2 = Time between HSI(.0) service and Software Timer 0 MODULE = DSTSWT/Software Timers
0602	Carry Flag Test Failed EREG2 = xxxx MODULE = SYS02/System Power-up
0702	Overflow flags would not set correctly EREG2 = 0002h MODULE = SYS02/System Power-up
0702	Overflow flags would not clear correctly EREG2 = xxxx MODULE = SYS02/System Power-up
0802	Interrupt Pending Register failed read/write test EREG2 = offending Interrupt Pending byte MODULE = SYS02/System Power-up
xx91	(user defined) EREG2 = (user defined) MODULE = DSTTOV/Timer Overflows

APPENDIX B

DIAG96.LIB Error Messages by Module Name

ALU01	Add/Subtract	0011 All Tests Passed ERE2 = 0000
		0111 An Addition error occurred ERE2 = offending argument when the error occurred
		0211 A Subtraction error occurred ERE2 = offending argument when the error occurred
		0311 A flag error occurred ERE2 = offending argument when the error occurred
ALU02	MULUB	0012 All Tests Passed ERE2 = 0000
		0112 Incorrect multiplication result was detected ERE2 = Multiplier/Multiplicand
ALU03	Multiply/Divide Table	0013 All Tests Passed ERE2 = 0000
		0115 A signed operation failed ERE2 = offending argument on error
		0215 An unsigned operation failed ERE2 = offending argument on error
		0315 A flag error occurred ERE2 = offending argument on error
ALU04	Multiply/Divide Random	0014 All Tests Passed ERE2 = 0000
		0115 A signed operation failed ERE2 = offending argument on error
		0215 An unsigned operation failed ERE2 = offending argument on error
		0315 A flag error occurred ERE2 = offending argument on error
ALU05	Multiply/Divide Core	0015 All Tests Passed ERE2 = 0000
		0115 A signed operation failed ERE2 = offending argument on error
		0215 An unsigned operation failed ERE2 = offending argument on error
		0315 A flag error occurred ERE2 = offending argument on error

D96A96	All Tests in ASM96 0030 All Tests Passed, checksum is ready ERE2 = 16-bit checksum
D96FST	Selected Tests in ASM 00E0 All Tests Passed, checksum is over range specified ERE2 = 16-bit checksum
D96P96	ALL Tests in PLM96 00F0 All Tests Passed, checksum is ready ERE2 = 16-bit checksum
DSTA2D	A/D Conversion Complete 01C0 An unexpected A/D conversion result was found ERE2 = Channel number of unexpected result
DSTEX1	External Interrupt (Supervisor) 01A0 One or more DST Module did not execute on time ERE2 = Number of SHIFTS done
DSTHSI	High Speed Inputs 0161 A high pulse on HSI.2 had an unexpected width ERE2 = difference between actual and expected pulse width 0261 A low pulse on HSI.2 had an unexpected width ERE2 = difference between actual and expected pulse width 0361 A high pulse on HSI.3 had an unexpected width ERE2 = difference between actual and expected pulse width 0461 A low pulse on HSI.3 had an unexpected width ERE2 = difference between actual and expected pulse width 0561 HSI unit indicated an HSI.1 event occurred ERE2 = Time recorded in HSI FIFO
DSTISR	DST Initialization 0040 Initialization completed satisfactorily ERE2 = 0000 0140 An abnormal RESET occurred ERE2 = TIMER1 0240 T2CLK will not change ERE2 = xxxx 0340 T2RST pin would not RESET T2CLK ERE2 = xxxx 0440 IOC0.1 would not RESET T2CLK ERE2 = xxxx
DSTSER	Serial Port 01B0 An unexpected serial character was received ERE2 = Bad character received 02B0 A carriage return was received out of sequence ERE2 = number of characters since a carriage return

DSTSWT	Software Timers	01D0 Found unexpected value on PORT1 ERE2 = expected value in high byte, actual in low byte
		01D1 Invalid T2CLK value reached ERE2 = T2CLK
		02D0 A/D Done did not occur within BURST window ERE2 = Time between A/D done and Software Timer 0
		02D1 Test reached an illegal Software Timer 0 state ERE2 = Illegal case jump made
		03D0 REAL TIME update did not occur within BURST window ERE2 = Time between REAL TIME update and Software Timer 0
		04D0 HSO.0 response did not occur within BURST window ERE2 = Time between HSO.0 update and Software Timer 0
		05D0 HSI(.0) response did not occur within BURST window ERE2 = Time between HSI(.0) service and Software Timer 0
DSTTOV	Timer Overflows	0190 An overflow of T2CLK was indicated ERE2 = TIMER1
		xx91 (user defined) ERE2 = (user defined)
MEM01	Complementary Address (Registers)	0021 All Tests Passed ERE2 = 0000
		0121 A memory location failed ERE2 = address of the error
MEM02	Walking Ones/Zeros (Registers)	0022 All Tests Passed ERE2 = 0000
		0122 A memory location failed ERE2 = address of the error
MEM03	Gallopings Ones/Zeros (Registers)	0023 All Tests Passed ERE2 = 0000
		0123 A memory location failed ERE2 = address of the error
MEM04	Bits Set	0024 No bits were set in the byte tested ERE2 = 0000
		0124 At least one bit was set in the byte tested ERE2 = number of bits set

MEM05	Checkerboard Pattern (Registers)	
	0025 All Tests Passed ERE2 = 0000	
	0125 A memory location failed	
	ERE2 = address of the error	
MEM06	Complementary Address	
	0026 All Tests Passed ERE2 = 0000	
	0126 A memory location failed	
	ERE2 = address of the error	
MEM07	Walking Ones	
	0027 All Tests Passed ERE2 = 0000	
	0127 A memory location failed	
	ERE2 = address of the error	
MEM08	Galloping Ones	
	0028 All Tests Passed ERE2 = 0000	
	0128 A memory location failed	
	ERE2 = address of the error	
MEM09	Walking Ones/Zeros	
	0029 All Tests Passed ERE2 = 0000	
	0129 A memory location failed	
	ERE2 = address of the error	
MEM0A	Galloping Ones/Zeros	
	002A All Tests Passed ERE2 = 0000	
	012A A memory location failed	
	ERE2 = address of the error	
MEM0B	Checksum	
	012B 16-bit Checksum is ready ERE2 = 16-bit Checksum	
MEM0C	User Pattern (Registers)	
	002C All Tests Passed ERE2 = 0000	
	012C A memory location failed	
	ERE2 = address of the error	
MEM0D	User Pattern	
	002D All Tests Passed ERE2 = 0000	
	012D A memory location failed	
	ERE2 = address of the error	

MCS®-96 Diagnostics Library

SYS01	Common Symbols	0000 No Message ERE2 = 0ffffh
SYS02	System Power-up	0002 All Tests Passed ERE2 = 0000h
	0102 I/O Status Registers were unexpected	ERE2 = IOS0 in low byte, IOS1 in high byte
	0202 TIMER1 did not change over time	ERE2 = TIMER1
	0302 Zero Register was found to change	ERE2 = Program Status Word At Failure
	0402 PUSHF or POPF failed	ERE2 = Erroneous PUSHed or POPed value found
	0502 Sticky Bit would not set	ERE2 = 3fffh
	0502 Sticky Bit would not clear	ERE2 = 0000
	0602 Carry Flag Test Failed	ERE2 = xxxx
	0702 Overflow flags would not set correctly	ERE2 = 0002h
	0702 Overflow flags would not clear correctly	ERE2 = xxxx
	0802 Interrupt Pending Register failed read/write test	ERE2 = offending Interrupt Pending byte
SYS03	Program Counter	0003 All Tests Passed ERE2 = 0000
	0103 Test Code Returned Early	ERE2 = Early Time
	0203 Test Code Returned Late	ERE2 = Late Time
	0303 Counter Register contained unexpected value	ERE2 = Erroneous Counter Value

APPENDIX C

DESCRIPTION OF DIAG96.LIB BATCH FILES

The batch files that come with the library will help speed the process of either linking to the library as is, or revising library programs to suit custom purposes.

Some batch files require a parameter that provides the extensionless name of a user definable variable file to be included in the action of the batch file.

All DIAG96.LIB batch files assume that both the source and destination files reside in the same directory. Given the size of the library, and the fact that all of the files will not fit on one floppy disk, the command files will need to be edited if the user's system is not equipped with a hard disk.

INSTALL.BAT — Used to install the library on a hard disk system. To install the library, create a directory called \DIAG96 under the main directory, insert disk 1 into drive a: and type:

a:Install

DST360K.BAT & DST12MEG.BAT — CAUTION: THESE BATCH FILES WILL FORMAT AND DESTROY ALL INFORMATION ON THE FLOPPIES USED.

These command files were created to make the DIAG96.LIB disk set. DST360K was created for use with 360K floppy disks and requires three diskettes. DST12MEG was created for use with 1.2M disks and only needs two diskettes. The batch files will prompt you to change disks. MAKE SURE TO ENTER THE CORRECT DISK DRIVE WHEN INVOKING THESE BATCH FILES. ALSO MAKE SURE TO INCLUDE THE DRIVE ID IN THE COMMAND LINE. THESE BATCH FILES FIRST FORMAT THE DISK, AND WE ALL KNOW WHAT WHEN DOS DEFAULTS TO THE HARD DISK!!!!!!!!!!

For example:

DST12MEG a:

SCRUB.BAT — CAUTION: THIS FILE DELETES FILES USING WILDCARDS.

All Diagnostic Library related files are deleted for the \DIAG96 directory. SYS?? and MEM?? wildcards are used, so be forewarned. This batch file does not delete itself!!!! To invoke this batch file, type:

Scrub

D96ASM.BAT — Assembles all General Diagnostic modules including the PLM compilation of D96P96.P96. To invoke the batch file, get in the \DIAG96 directory and type:

D96ASM

DSTASM.BAT — Assembles all Dynamic Stability Test modules. To invoke the batch file, get in \DIAG96 directory and type:

DSTASM

D96LP.BAT — Copies all General Diagnostic list files to a printer. Invocation must include a device where the printer resides. For example:

D96LP lpt1

DSTLP.BAT — Copies all Dynamic Stability Test modules to a printer. Invocation must include a device where the printer resides. For example:

DSTLP.BAT lpt1

LPONLY.BAT — Executes D96LP.BAT and DSTLP.BAT. Invocation must include a device where the printer resides. For example:

LPONLY lpt1

MCS®-96 Diagnostics Library

D96LIB.BAT — Deletes the current DIAG96.LIB collection. Also creates a new library of the same name using the files resident in the \DIAG96 directory bearing the **General Diagnostics** names. The DST96.LIB is not altered, and is included in the new DIAG96.LIB. To invoke the batch file, get in the \DIAG96 directory and type:

D96LIB

DSTLIB.BAT — Deletes the current DST96.LIB collection. Also creates a new library of the same name using the files resident in the \DIAG96 directory bearing the **Dynamic Stability Test** names. Since DST96.LIB is included in DIAG96.LIB, DIAG96.LIB is recreated by an invocation of D96LIB.BAT. To invoke this batch file, get in the \DIAG96 directory and type:

DSTLIB

DSTRL.BAT — This batch file is of most interest to **Dynamic Stability Test** users. It links a specified main task to the library. This file makes assumptions about the hardware memory implementation that may not be correct. Therefore minor changes may need to be made to the DSTRL.BAT RL96 invocation statement. A file name without extension must be provided and that file must reside in the \DIAG96 directory. The batch file assumes that the extension of the object file to be linked to the library is .OBJ. For example:

DSTRL Example_task

BLASTP.BAT — This batch file assembles the specified input file, then executes D96ASM.BAT, DSTASM.BAT, LPONLY.BAT, DSTLIB.BAT, and DSTRL.BAT. Then, the listfile output of the user's assembly and the print file of the linkage are copied to the printer specified. The batch file assumes that the input file is in the \DIAG96 directory and has a .A96 extension. For example:

BLASTP Example_lpt1

BLASTN.BAT — This batch file executes all assemblies, compilations, and linkages executed in BLASTP.BAT, but no copies are sent to the printer. The batch file assumes that the input file is in the \DIAG96 directory and has a .A96 extension. For example:

BLASTN Example_task

REGEN.BAT — Used to regenerate the library when only one module has changed. Specify the module that has changed when invoking this batch file. For example:

REGEN ALU03

MAKPLM.BAT — Used to make an impostor PLM96.LIB. The library created is not a real PLM96.LIB, and will not work with PLM programs. However, it is what is needed to use DIAG96.LIB. To invoke this batch file, get in the \DIAG96 director and type:

MAKPLM

MAKBH.BAT — Used to modify the library to run in an 8X9XBH. The 8X9XBH fails a flag test because of the -90 bug relating to the Z flag on add and subtract with carry is inadvertently verified by a library test. To invoke this batch file, get in the \DIAG96 directory and type:

MAKBH

D96RL.BAT — A generalized command that links target modules to DIAG96.LIB. It is intended for used when only the **General Diagnostics** are being used. Provide the target object file name and the directory in which it resides. For example:

D96RL \SOURCE\Example_

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F5:D96A96.A96

OBJECT FILE: F5:D96A96.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOGEN DEBUG

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			1	
			2	*****
			3	ALL TESTS ASM96 MODULE STACKSIZE(20)
			4	***** 0030
			5	
			6	in order to run this module, the STACK must be ALL external, and the
			7	data ram partitioned for memory test must not include ANY of the STACK
			8	
			9	To call this module
			10	
			11	PUSH #<RAM segment1 start address>
			12	PUSH #<RAM segment1 ending address>
			13	PUSH #<RAM segment2 start address>
			14	PUSH #<RAM segment2 ending address>
			15	PUSH #<random seed>
			16	PUSH #<number of cycles desired for random test>
			17	PUSH #<address of the last byte of rom>
			18	PUSH #<an argument for mul/div tests>
			19	PUSH #<a second argument for mul/div tests>
			20	PUSH #<a bit pattern for memory tests>
			21	CALL D96A96
			22	
			23	
			24	Remember, this test will take a long time if large memory regions are
			25	partitioned, or if a large number of cycles of random test numbers is
			26	requested. For example, with 8Kbytes of Ram in each region the test
			27	executes in 3 hours.
			28	
			29	It is suggested that for large memory tests, that the complimentary
			30	address test be done on the whole region at once. Then, the more
			31	exhaustive tests done on each memory chip in the system.
			32	*****
			33	
0000			34	rseg
			35	
			36	extrn sp, ereq1, ereq2
			37	
			38	
3000			39	cseg at 3000h
			40	extrn sys01, sys02, sys03, alu01, alu02, alu03, alu04, alu05
			41	extrn mem01, mem02, mem03, mem04, mem05, mem06, mem07, mem08
			42	extrn mem09, mem0a, mem0b, mem0c, mem0d
			43	
			44	PUBLIC D96A96
			45	\$eject
			46	
			47	\$include(:f3:dstdmac.inc) ;provides the macro BR ON Error
=1			48	*****
=1			49	;DST Macros INCLUDE FILE ;*****
=1			50	*****
=1			51	

Example Program Listings

APPENDIX D

D96A96

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		104	
		105	
0000		106	D96A96: INCLUDE BITE
		107	
0000 EF0000	E	108	CALL sys02 ;CALL the System Power Up Test
		109	
		110	BR_ON_ERR Error_Found
		114	
000A EF0000	E	115	CALL alu01 ;CALL the Add/Subtract test
		116	
		117	BR_ON_ERR Error_Found
		121	
0014 EF0000	E	122	CALL alu02 ;CALL the MULUB test
		123	
		124	BR_ON_ERR Error_Found
		128	
001E EF0000	E	129	CALL alu03 ;CALL the Multiply/Divide Table
		130	;driven test
		131	BR_ON_ERR Error_Found
		135	
0028 CB000C	E	136	PUSH 0ch[sp] ;PUSH a random seed
002B CB000C	E	137	PUSH 0ch[sp] ;PUSH the number of tests desired
002E EF0000	E	138	CALL alu04 ;CALL the Multiply/Divide Random test
		139	
		140	BR_ON_ERR Error_Found
		144	
0038 CB0006	E	145	PUSH 06h[sp] ;PUSH an argument
003B CB0006	E	146	PUSH 06h[sp] ;PUSH another argument
003E EF0000	E	147	CALL alu05 ;CALL the Multiply/Divide Core Test
		148	
		149	BR_ON_ERR Error_Found
		153	
0048 EF0000	E	154	CALL mem01 ;CALL a Complementary Address test
		155	;on the internal registers
		156	BR_ON_ERR Error_Found
		160	
0052 EF0000	E	161	CALL mem02 ;CALL a Walking 1s/0s test on
		162	;the internal registers
		163	BR_ON_ERR Error_Found
		167	
005C EF0000	E	168	CALL mem03 ;CALL a Galloping 1s/0s test on
		169	;the internal registers
		170	BR_ON_ERR Error_Found
		174	
0066 C800	E	175	PUSH zero ;PUSH a zero
0068 EF0000	E	176	CALL mem04 ;CALL the Bits Set Test
		177	
		178	BR_ON_ERR Error_Found
		182	
0072 EF0000	E	183	CALL mem05 ;CALL a Checkerboard Pattern test
		184	;for internal registers
		185	BR_ON_ERR Error_Found
		189	
007C CB0014	E	190	PUSH 14h[sp] ;PUSH the start address
007F CB0014	E	191	PUSH 14h[sp] ; and the end address of a RAM area
0082 EF0000	E	192	CALL mem06 ; and CALL a Complementary Address test
		193	

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			194	BR_ON_ERR Error_Found
			198	
	008C	CB0010	E 199	PUSH 10h[sp] ;PUSH a second start and end address
	008F	CB0010	E 200	PUSH 10h[sp] ; and repeat the
	0092	EF0000	E 201	CALL mem06 ;Complementary Address test
			202	
			203	BR_ON_ERR Error_Found
			207	
	009C	CB0014	E 208	PUSH 14h[sp] ;PUSH a start address
	009F	CB0014	E 209	PUSH 14h[sp] ;PUSH an ending address
	00A2	EF0000	E 210	CALL mem07 ;CALL a Walking Ones test
			211	
			212	BR_ON_ERR Error_Found
			216	
	00AC	CB0010	E 217	PUSH 10h[sp] ;PUSH the start and end address
	00AF	CB0010	E 218	PUSH 10h[sp] ; for another section of RAM
	00B2	EF0000	E 219	CALL mem07 ; and repeat the Walking Ones test
			220	
			221	BR_ON_ERR Error_Found
			225	
	00BC	CB0014	E 226	PUSH 14h[sp] ;PUSH a start address
	00BF	CB0014	E 227	PUSH 14h[sp] ;PUSH an ending address
	00C2	EF0000	E 228	CALL mem08 ;CALL a Galloping Ones test
			229	
			230	BR_ON_ERR Error_Found
			234	
			235	
	00CC	CB0010	E 236	PUSH 10h[sp] ;PUSH a second start and end address
	00CF	CB0010	E 237	PUSH 10h[sp] ; for another region of RAM and
	00D2	EF0000	E 238	CALL mem08 ;CALL the Galloping Ones test again
			239	
			240	BR_ON_ERR Error_Found
			244	
	00DC	CB0014	E 245	PUSH 14h[sp] ;PUSH the start and end address of
	00DF	CB0014	E 246	PUSH 14h[sp] ; a region of RAM
	00E2	EF0000	E 247	CALL mem09 ;CALL the Walking 1s/0s test
			248	
			249	BR_ON_ERR Error_Found
			253	
			254	
	00EC	CB0010	E 255	PUSH 10h[sp] ;PUSH the start and end address of
	00EF	CB0010	E 256	PUSH 10h[sp] ; another region of RAM
	00F2	EF0000	E 257	CALL mem09 ;CALL the Walking 1s/0s test again
			258	
			259	BR_ON_ERR Error_Found
			263	
	00FC	CB0014	E 264	PUSH 14h[sp] ;PUSH the start and end address of
	00FF	CB0014	E 265	PUSH 14h[sp] ; a region of RAM
	0102	EF0000	E 266	CALL mem0a ;CALL a Galloping 1s/0s test
			267	
			268	BR_ON_ERR Error_Found
			272	
			273	
	010C	CB0010	E 274	PUSH 10h[sp] ;PUSH the start and end address of
	010F	CB0010	E 275	PUSH 10h[sp] ; another region of RAM
	0112	EF0000	E 276	CALL mem0a ;CALL the Galloping 1s/0s test again
			277	

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT	
			278	BR_ON_ERR	Error_Found
			282		
			283		
011A	CB0002	E	284	PUSH	02h[sp]
011D	EF0000	E	285	CALL	mem0c
			286		
			287	BR_ON_ERR	Error_Found
			291		
0125	CB0014	E	292	PUSH	14h[sp]
0128	CB0014	E	293	PUSH	14h[sp]
012B	CB0006	E	294	PUSH	06h[sp]
012E	EF0000	E	295	CALL	mem0d
			296		
			297	BR_ON_ERR	Error_Found
			301		
0136	CB0010	E	302	PUSH	10h[sp]
0139	CB0010	E	303	PUSH	10h[sp]
013C	CB0006	E	304	PUSH	06h[sp]
013F	EF0000	E	305	CALL	mem0d
			306		
			307	BR_ON_ERR	Error_Found
			311		
0147	CB0014	E	312	PUSH	14h[sp]
014A	CB0014	E	313	PUSH	14h[sp]
014D	EF0000	E	314	CALL	sys03
			315		
			316	BR_ON_ERR	Error_Found
			320		
0155	CB0010	E	321	PUSH	10h[sp]
0158	CB0010	E	322	PUSH	10h[sp]
015B	EF0000	E	323	CALL	sys03
			324		
			325	BR_ON_ERR	Error_Found
			329		
0163	C98020	E	330	PUSH	#2080h
0166	CB000A	E	331	PUSH	0ah[sp]
0169	EF0000	E	332	CALL	mem0b
			333		
016C	A1300000	E	334	LD	EREG1, #0030h
			335		
			336		
0170	CF0014	E	337	POP	14h[sp]
0173	65120000	E	338	ADD	sp, #12h
0177	F0	E	339	RET	
			340		
0178	CB0014	E	341	Error_Found:	
			342		
0178	CF0014	E	343	POP	14h[sp]
017B	65120000	E	344	ADD	sp, #12h
017F	F0	E	345	RET	
			346		
0180	CB0014	E	347	end	

SYMBOL TABLE LISTING

N A M E	VALUE	ATTRIBUTES
ALL TESTS_ASM96	-----	MODULE STACKSIZE(20)
ALU01	-----	CODE EXTERNAL
ALU02	-----	CODE EXTERNAL
ALU03	-----	CODE EXTERNAL
ALU04	-----	CODE EXTERNAL
ALU05	-----	CODE EXTERNAL
BR ON ERR	-----	MACRO
D96A96	0000H	CODE REL PUBLIC ENTRY
EREG1	-----	REG EXTERNAL
EREG2	-----	REG EXTERNAL
ERROR FOUND	0178H	CODE REL ENTRY
MACRO TEMP	0000H	REG REL BYTE
MEM01	-----	CODE EXTERNAL
MEM02	-----	CODE EXTERNAL
MEM03	-----	CODE EXTERNAL
MEM04	-----	CODE EXTERNAL
MEM05	-----	CODE EXTERNAL
MEM06	-----	CODE EXTERNAL
MEM07	-----	CODE EXTERNAL
MEM08	-----	CODE EXTERNAL
MEM09	-----	CODE EXTERNAL
MEM0A	-----	CODE EXTERNAL
MEM0B	-----	CODE EXTERNAL
MEM0C	-----	CODE EXTERNAL
MEM0D	-----	CODE EXTERNAL
RESET WATCHDOG	-----	MACRO
RI	0006H	NULL ABS
SP	-----	REG EXTERNAL
SP STAT	-----	REG EXTERNAL
SPSTATUS	-----	MACRO
SPWAIT	-----	MACRO
SYS01	-----	CODE EXTERNAL
SYS02	-----	CODE EXTERNAL
SYS03	-----	CODE EXTERNAL
TI	0005H	NULL ABS
ZERO	-----	REG EXTERNAL

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

DO1
W112D176D02100001

COMPILES TRACKED BY: BFW02 30 13:030000 500 CODE DEMO
ORIGINAL SOURCE BYCND IN 113100000 001
SERIES-111 BFW-00 A1'S COMB 05 REDDIT WTDIV00000000

SERIES-III PL/M-96 V1.0 COMPILATION OF MODULE ALLDIAG96TESTS
 OBJECT MODULE PLACED IN :F2:D96P96.OBJ
 COMPILER INVOKED BY: PLM96.86 :F2:D96P96.P96 CODE DEBUG

```

1      All$Ddiag96$Tests:
      DO;

2 1      SYS02: PROCEDURE DWORD EXTERNAL;
3 2      END SYS02;

4 1      SYS03: PROCEDURE (parml,parm2) DWORD EXTERNAL;
5 2      DECLARE (parml,parm2) WORD;
6 2      END SYS03;

7 1      ALU01: PROCEDURE DWORD EXTERNAL;
8 2      END ALU01;

9 1      ALU02: PROCEDURE DWORD EXTERNAL;
10 2      END ALU02;

11 1      ALU03: PROCEDURE DWORD EXTERNAL;
12 2      END ALU03;

13 1      ALU04: PROCEDURE (parml,parm2) DWORD EXTERNAL;
14 2      DECLARE (parml,parm2) WORD;
15 2      END ALU04;

16 1      ALU05: PROCEDURE (parml,parm2) DWORD EXTERNAL;
17 2      DECLARE (parml,parm2) WORD;
18 2      END ALU05;

19 1      MEM01: PROCEDURE DWORD EXTERNAL;
20 2      END MEM01;

21 1      MEM02: PROCEDURE DWORD EXTERNAL;
22 2      END MEM02;

23 1      MEM03: PROCEDURE DWORD EXTERNAL;
24 2      END MEM03;

25 1      MEM04: PROCEDURE (parml) DWORD EXTERNAL;
26 2      DECLARE (parml) WORD;
27 2      END MEM04;

28 1      MEM05: PROCEDURE DWORD EXTERNAL;
29 2      END MEM05;

30 1      MEM06: PROCEDURE (parml,parm2) DWORD EXTERNAL;
31 2      DECLARE (parml,parm2) WORD;
32 2      END MEM06;

33 1      MEM07: PROCEDURE (parml,parm2) DWORD EXTERNAL;
34 2      DECLARE (parml,parm2) WORD;
35 2      END MEM07;

```



```

136 1      MEM08: PROCEDURE (parm1,parm2) DWORD EXTERNAL;
137 2      DECLARE (parm1,parm2) WORD;
138 2      END MEM08;
139 1      MEM09: PROCEDURE (parm1,parm2) DWORD EXTERNAL;
140 2      DECLARE (parm1,parm2) WORD;
141 2      END MEM09;

142 1      MEM0A: PROCEDURE (parm1,parm2) DWORD EXTERNAL;
143 2      DECLARE (parm1,parm2) WORD;
144 2      END MEM0A;

145 1      MEM0B: PROCEDURE (parm1,parm2) DWORD EXTERNAL;
146 2      DECLARE (parm1,parm2) WORD;
147 2      END MEM0B;

148 1      MEM0C: PROCEDURE (parm1) DWORD EXTERNAL;
149 2      DECLARE (parm1) WORD;
150 2      END MEM0C;

151 1      MEM0D: PROCEDURE (parm1,parm2,parm3) DWORD EXTERNAL;
152 2      DECLARE (parm1,parm2,parm3) WORD;
153 2      END MEM0D;

154 1      DECLARE result DWORD;
155 1      DECLARE error$codes STRUCTURE (number WORD,detail WORD) AT (.result);

156 1      D96P96: PROCEDURE (ram1$start,ram1$stop,
157 2      ram2$start,ram2$stop,
158 2      random$seed,random$length,
159 2      top$of$code,
160 2      argument1,argument2,
161 2      bit$pattern) DWORD PUBLIC;

162 2      DECLARE (ram1$start,ram1$stop,
163 2      ram2$start,ram2$stop,
164 2      random$seed,random$length,
165 2      top$of$code,
166 2      argument1,argument2,
167 2      bit$pattern) WORD SLOW;

168 2      result=SYS02;
169 2      IF error$codes.number > 255 THEN GOTO end$tests;

170 2      result=ALU01;
171 2      IF error$codes.number > 255 THEN GOTO end$tests;

172 2      result=ALU02;
173 2      IF error$codes.number > 255 THEN GOTO end$tests;

174 2      result=ALU03;
175 2      IF error$codes.number > 255 THEN GOTO end$tests;

176 2      result=ALU04(47efH,1000H);

```

```

71 2      IF error$codes.number > 255 THEN GOTO end$tests;
73 2      result=ALU05(argument1,argument2);
74 2      IF error$codes.number > 255 THEN GOTO end$tests;
76 2      result=MEM01;
77 2      IF error$codes.number > 255 THEN GOTO end$tests;
79 2      result=MEM02;
80 2      IF error$codes.number > 255 THEN GOTO end$tests;
82 2      result=MEM03;
83 2      IF error$codes.number > 255 THEN GOTO end$tests;
85 2      result=MEM04(0);
86 2      IF error$codes.number > 255 THEN GOTO end$tests;
88 2      result=MEM05;
89 2      IF error$codes.number > 255 THEN GOTO end$tests;
91 2      result=MEM06(ram1$start,ram1$stop);
92 2      IF error$codes.number > 255 THEN GOTO end$tests;
94 2      result=MEM06(ram2$start,ram2$stop);
95 2      IF error$codes.number > 255 THEN GOTO end$tests;
97 2      result=MEM07(ram1$start,ram1$stop);
98 2      IF error$codes.number > 255 THEN GOTO end$tests;
100 2     result=MEM07(ram2$start,ram2$stop);
101 2     IF error$codes.number > 255 THEN GOTO end$tests;
103 2     result=MEM08(ram1$start,ram1$stop);
104 2     IF error$codes.number > 255 THEN GOTO end$tests;
106 2     result=MEM08(ram2$start,ram2$stop);
107 2     IF error$codes.number > 255 THEN GOTO end$tests;
109 2     result=MEM09(ram1$start,ram1$stop);
110 2     IF error$codes.number > 255 THEN GOTO end$tests;
112 2     result=MEM09(ram2$start,ram2$stop);
113 2     IF error$codes.number > 255 THEN GOTO end$tests;
115 2     result=MEM0A(ram1$start,ram1$stop);
116 2     IF error$codes.number > 255 THEN GOTO end$tests;
118 2     result=MEM0A(ram2$start,ram2$stop);
119 2     IF error$codes.number > 255 THEN GOTO end$tests;
121 2     result=MEM0C(bit$pattern);
122 2     IF error$codes.number > 255 THEN GOTO end$tests;
124 2     result=MEM0D(ram1$start,ram1$stop,bit$pattern);
125 2     IF error$codes.number > 255 THEN GOTO end$tests;
127 2     result=MEM0D(ram2$start,ram2$stop,bit$pattern);

```

```

0020 3100      1 21VLEWML 13
0020 3100      BE ENDLE212
0020 3100      1 21VLEWML 13
0020 3100      BIN 00002
0020 3100      CHB ENKOCODES'0000
0020 3100      1 21VLEWML 13
0020 3100      FD RES01'JM58
0020 3100      FD RES01+3H'JM53
0020 3100      CVTT VT004
0020 3100      B02H 17000
0020 3100      B02H 04100
0020 3100      00001:
0020 3100      1 21VLEWML 10
0020 3100      BE ENDLE212
0020 3100      1 21VLEWML 00
0020 3100      BIN 00001
0020 3100      CHB ENKOCODES'0000
128 2 IF error$codes.number > 255 THEN GOTO end$tests;
0020 3100      FD RES01'JM58
130 2 result=SYS03(ram1$start,ram1$stop);
0020 3100      FD RES01+3H'JM53
131 2 IF error$codes.number > 255 THEN GOTO end$tests;
0020 3100      CVTT VT003
133 2 result=SYS03(ram2$start,ram2$stop);
0020 3100      B02H 17000
134 2 IF error$codes.number > 255 THEN GOTO end$tests;
0020 3100      B02H 04100
136 2 result=MEM0B(2080h,top$of$code);
0020 3100      CHB ENKOCODES'0000
137 2 error$codes.number=00f0h;
0020 3100      FD RES01'JM58
138 2 end$tests: RETURN result;
0020 3100      FD RES01+3H'JM53
139 2 END D96P96;
0020 3100      CVTT VT003
140 1 312 END All$Ddiag96$Tests;
0020 3100      BE ENDLE212
0020 3100      1 21VLEWML 03
0020 3100      BIN 00003
0020 3100      CHB ENKOCODES'0000
0020 3100      1 21VLEWML 03
0020 3100      FD RES01'JM58
0020 3100      FD RES01+3H'JM53
0020 3100      CVTT VT001
0020 3100      00001:
0020 3100      1 21VLEWML 01
0020 3100      BE ENDLE212
0020 3100      1 21VLEWML 00
0020 3100      BIN 00001
0020 3100      CHB ENKOCODES'0000
0020 3100      1 21VLEWML 03
0020 3100      FD RES01'JM58
0020 3100      FD RES01+3H'JM53
0020 3100      CVTT VT003
0020 3100      1 21VLEWML 00
0020 3100      BE ENDLE212
0020 3100      1 21VLEWML 00
0020 3100      BIN 00001
0020 3100      CHB ENKOCODES'0000
0020 3100      1 21VLEWML 03
0020 3100      FD RES01'JM58
0020 3100      FD RES01+3H'JM53
0020 3100      CVTT VT001
0020 3100      00001:
0020 3100      1 21VLEWML 00

```

```

; STATEMENT 56
D96P96:
0000 C800 E PUSH ?FRAME01
0002 A01800 E LD ?FRAME01,SP
; STATEMENT 58
0005 EF0000 E CALL SYS02
0008 A01E02 R LD RESULT+2H,TMP2
000B A01C00 R LD RESULT,TMP0
; STATEMENT 59
000E 89FF0000 R CMP ERRORCODES,#0FFH
0012 D102 BNH @0001
; STATEMENT 60
0014 2226 BR ENDTESTS
; STATEMENT 61
@0001:
0016 EF0000 E CALL ALU01
0019 A01E02 R LD RESULT+2H,TMP2
001C A01C00 R LD RESULT,TMP0
; STATEMENT 62
001F 89FF0000 R CMP ERRORCODES,#0FFH
0023 D102 BNH @0002
; STATEMENT 63
0025 2215 BR ENDTESTS
; STATEMENT 64
@0002:
0027 EF0000 E CALL ALU02
002A A01E02 R LD RESULT+2H,TMP2
002D A01C00 R LD RESULT,TMP0
; STATEMENT 65
0030 89FF0000 R CMP ERRORCODES,#0FFH
0034 D102 BNH @0003
; STATEMENT 66
0036 2204 BR ENDTESTS
; STATEMENT 67
@0003:
0038 EF0000 E CALL ALU03
003B A01E02 R LD RESULT+2H,TMP2
003E A01C00 R LD RESULT,TMP0
; STATEMENT 68
0041 89FF0000 R CMP ERRORCODES,#0FFH
0045 D102 BNH @0004
; STATEMENT 69
0047 21F3 BR ENDTESTS
; STATEMENT 70
@0004:
0049 C9EF47 PUSH #47EFH
004C C90010 PUSH #1000H
004F EF0000 E CALL ALU04
0052 A01E02 R LD RESULT+2H,TMP2
0055 A01C00 R LD RESULT,TMP0
; STATEMENT 71
0058 89FF0000 R CMP ERRORCODES,#0FFH
005C D102 BNH @0005
; STATEMENT 72
005E 21DC BR ENDTESTS
; STATEMENT 73

```



```

0060      CB0008      E      @0005:      PUSH  ARGUMENT1[?FRAME01]
0063      CB0006      E      PUSH  ARGUMENT2[?FRAME01]
0066      EF0000      E      CALL  ALU05
0069      A01E02      R      LD      RESULT+2H,TMP2
006C      A01C00      R      LD      RESULT,TMP0
006F      89FF0000    R      CMP     ERRORCODES,#0FFH
0073      D10200      R      BNH     @0006
0075      21C500      E      ;      STATEMENT 75
0077      CB0010      E      BR      ENDTESTS
0077      EF0000      E      @0006:      STATEMENT 76
007A      A01E02      R      CALL  MEM01
007D      A01C00      R      LD      RESULT+2H,TMP2
0080      89FF0000    R      LD      RESULT,TMP0
0084      D10200      R      ;      STATEMENT 77
0086      21B400      R      CMP     ERRORCODES,#0FFH
0088      EF0000      E      BNH     @0007
0088      A01E02      R      ;      STATEMENT 78
008E      A01C00      R      BR      ENDTESTS
0091      89FF0000    R      @0007:      STATEMENT 79
0095      D10200      R      CALL  MEM02
0097      21A300      R      LD      RESULT+2H,TMP2
0099      EF0000      E      LD      RESULT,TMP0
0099      A01E02      R      ;      STATEMENT 80
009C      A01C00      R      CMP     ERRORCODES,#0FFH
009F      A01C00      R      BNH     @0008
00A2      89FF0000    R      ;      STATEMENT 81
00A6      D10200      R      BR      ENDTESTS
00A8      219200      R      @0008:      STATEMENT 82
00AA      C80000      E      CALL  MEM03
00AC      EF0000      E      LD      RESULT+2H,TMP2
00AF      A01E02      R      LD      RESULT,TMP0
00B2      A01C00      R      ;      STATEMENT 83
00B5      89FF0000    R      CMP     ERRORCODES,#0FFH
00B9      D10200      R      BNH     @0009
00BB      217F00      R      ;      STATEMENT 84
00BD      EF0000      E      BR      ENDTESTS
00BD      A01E02      E      @0009:      STATEMENT 85
00C0      A01C00      R      CALL  MEM04
00C0      A01E02      R      LD      RESULT+2H,TMP2
00C0      A01E02      R      LD      RESULT,TMP0
00C0      A01E02      R      ;      STATEMENT 86
00C0      A01E02      R      CMP     ERRORCODES,#0FFH
00C0      A01E02      R      BNH     @000A
00C0      A01E02      R      ;      STATEMENT 87
00C0      A01E02      R      BR      ENDTESTS
00C0      A01E02      R      @000A:      STATEMENT 88
00C0      A01E02      R      CALL  MEM05
00C0      A01E02      R      LD      RESULT+2H,TMP2
00C0      A01E02      R      LD      RESULT,TMP0

```

```

00C3 A01C00      R      LD      RESULT,TMP0
00C6 89FF0000    R      CMP     ERRORCODES,#0FFH
00CA D102        R      BNH     @000B
; STATEMENT      90
00CC 216E        R      BR      ENDTESTS
; STATEMENT      91
00CE CB0016      E @000B: PUSH  RAM1START[?FRAME01]
00D1 CB0014      E      PUSH  RAM1STOP[?FRAME01]
00D4 EF0000      E      CALL  MEM06
00D7 A01E02      R      LD      RESULT+2H,TMP2
00DA A01C00      R      LD      RESULT,TMP0
; STATEMENT      92
00DD 89FF0000    R      CMP     ERRORCODES,#0FFH
00E1 D102        R      BNH     @000C
; STATEMENT      93
00E3 2157        R      BR      ENDTESTS
; STATEMENT      94
00E5 CB0012      E @000C: PUSH  RAM2START[?FRAME01]
00E8 CB0010      E      PUSH  RAM2STOP[?FRAME01]
00EB EF0000      E      CALL  MEM06
00EE A01E02      R      LD      RESULT+2H,TMP2
00F1 A01C00      R      LD      RESULT,TMP0
; STATEMENT      95
00F4 89FF0000    R      CMP     ERRORCODES,#0FFH
00F8 D102        R      BNH     @000D
; STATEMENT      96
00FA 2140        R      BR      ENDTESTS
; STATEMENT      97
00FC CB0016      E @000D: PUSH  RAM1START[?FRAME01]
00FF CB0014      E      PUSH  RAM1STOP[?FRAME01]
0102 EF0000      E      CALL  MEM07
0105 A01E02      R      LD      RESULT+2H,TMP2
0108 A01C00      R      LD      RESULT,TMP0
; STATEMENT      98
010B 89FF0000    R      CMP     ERRORCODES,#0FFH
010F D102        R      BNH     @000E
; STATEMENT      99
0111 2129        R      BR      ENDTESTS
; STATEMENT     100
0113 CB0012      E @000E: PUSH  RAM2START[?FRAME01]
0116 CB0010      E      PUSH  RAM2STOP[?FRAME01]
0119 EF0000      E      CALL  MEM07
011C A01E02      R      LD      RESULT+2H,TMP2
011F A01C00      R      LD      RESULT,TMP0
; STATEMENT     101
0122 89FF0000    R      CMP     ERRORCODES,#0FFH
0126 D102        R      BNH     @000F
; STATEMENT     102
0128 2112        R      BR      ENDTESTS
; STATEMENT     103
012A CB0010      E @000F: PUSH  RAM1START[?FRAME01]

```

```

012A CB0016      E      PUSH   RAM1START[?FRAME01]
012D CB0014      E      PUSH   RAM1STOP[?FRAME01]
0130 EF0000      E      CALL   MEM08
0133 A01E02      R      LD      RESULT+2H,TMP2
0136 A01C00      R      LD      RESULT,TMP0
                        ; STATEMENT 104
0139 89FF0000    R      CMP     ERRORCODES,#0FFH
013D D102        BNH     @0010
                        ; STATEMENT 105
013F 20FB        B      BR      ENDTESTS
                        ; STATEMENT 106
@0010:
0141 CB0012      E      PUSH   RAM2START[?FRAME01]
0144 CB0010      E      PUSH   RAM2STOP[?FRAME01]
0147 EF0000      E      CALL   MEM08
014A A01E02      R      LD      RESULT+2H,TMP2
014D A01C00      R      LD      RESULT,TMP0
                        ; STATEMENT 107
0150 89FF0000    R      CMP     ERRORCODES,#0FFH
0154 D102        BNH     @0011
                        ; STATEMENT 108
0156 20E4        B      BR      ENDTESTS
                        ; STATEMENT 109
@0011:
0158 CB0016      E      PUSH   RAM1START[?FRAME01]
015B CB0014      E      PUSH   RAM1STOP[?FRAME01]
015E EF0000      E      CALL   MEM09
0161 A01E02      R      LD      RESULT+2H,TMP2
0164 A01C00      R      LD      RESULT,TMP0
                        ; STATEMENT 110
0167 89FF0000    R      CMP     ERRORCODES,#0FFH
016B D102        BNH     @0012
                        ; STATEMENT 111
016D 20CD        B      BR      ENDTESTS
                        ; STATEMENT 112
@0012:
016F CB0012      E      PUSH   RAM2START[?FRAME01]
0172 CB0010      E      PUSH   RAM2STOP[?FRAME01]
0175 EF0000      E      CALL   MEM09
0178 A01E02      R      LD      RESULT+2H,TMP2
017B A01C00      R      LD      RESULT,TMP0
                        ; STATEMENT 113
017E 89FF0000    R      CMP     ERRORCODES,#0FFH
0182 D102        BNH     @0013
                        ; STATEMENT 114
0184 20B6        B      BR      ENDTESTS
                        ; STATEMENT 115
@0013:
0186 CB0016      E      PUSH   RAM1START[?FRAME01]
0189 CB0014      E      PUSH   RAM1STOP[?FRAME01]
018C EF0000      E      CALL   MEM0A
018F A01E02      R      LD      RESULT+2H,TMP2
0192 A01C00      R      LD      RESULT,TMP0
                        ; STATEMENT 116
0195 89FF0000    R      CMP     ERRORCODES,#0FFH
0199 D102        BNH     @0014

```

```

; STATEMENT 117
019B 209F BR ENDTESTS
; STATEMENT 118
019D @0014:
019D E PUSH RAM2START[?FRAME01]
01A0 E PUSH RAM2STOP[?FRAME01]
01A3 E CALL MEM0A
01A6 R LD RESULT+2H,TMP2
01A9 R LD RESULT,TMP0
; STATEMENT 119
01AC R CMP ERRORCODES,#0FFH
01B0 BNH @0015
; STATEMENT 120
01B2 BR ENDTESTS
01B2 ; STATEMENT 121
01B4 @0015:
01B4 E PUSH BITPATTERN[?FRAME01]
01B7 E CALL MEM0C
01BA R LD RESULT+2H,TMP2
01BD R LD RESULT,TMP0
; STATEMENT 122
01C0 R CMP ERRORCODES,#0FFH
01C4 BNH @0016
; STATEMENT 123
01C6 BR ENDTESTS
01C6 ; STATEMENT 124
01C8 @0016:
01C8 E PUSH RAM1START[?FRAME01]
01CB E PUSH RAM1STOP[?FRAME01]
01CE E PUSH BITPATTERN[?FRAME01]
01D1 E CALL MEM0D
01D4 R LD RESULT+2H,TMP2
01D7 R LD RESULT,TMP0
; STATEMENT 125
01DA R CMP ERRORCODES,#0FFH
01DE BNH @0017
; STATEMENT 126
01E0 BR ENDTESTS
01E0 ; STATEMENT 127
01E2 @0017:
01E2 E PUSH RAM2START[?FRAME01]
01E5 E PUSH RAM2STOP[?FRAME01]
01E8 E PUSH BITPATTERN[?FRAME01]
01EB E CALL MEM0D
01EE R LD RESULT+2H,TMP2
01F1 R LD RESULT,TMP0
; STATEMENT 128
01F4 R CMP ERRORCODES,#0FFH
01F8 BNH @0018
; STATEMENT 129
01FA BR ENDTESTS
01FA ; STATEMENT 130
01FC @0018:
01FC E PUSH RAM1START[?FRAME01]
01FF E PUSH RAM1STOP[?FRAME01]
0202 E CALL SYS03

```



```

0205 A01E02      R      LD      RESULT+2H,TMP2
0208 A01C00      R      LD      RESULT,TMP0
                        ;      STATEMENT 131
020B 89FF0000    R      CMP      ERRORCODES,#0FFH
020F D102        BNH      @0019
                        ;      STATEMENT 132
0211 2029        BR       ENDTTESTS
                        ;      STATEMENT 133
0213 @0019:      E      PUSH     RAM2START[?FRAME01]
0213 CB0012      E      PUSH     RAM2STOP[?FRAME01]
0216 CB0010      E      CALL     SYS03
0219 EF0000      R      LD      RESULT+2H,TMP2
021C A01E02      R      LD      RESULT,TMP0
021F A01C00      R      ;      STATEMENT 134
0222 89FF0000    R      CMP      ERRORCODES,#0FFH
0226 D102        BNH      @001A
                        ;      STATEMENT 135
0228 2012        BR       ENDTTESTS
                        ;      STATEMENT 136
022A @001A:      E      PUSH     #2080H
022A C98020      E      PUSH     TOPOFCODE[?FRAME01]
022D CB000A      E      CALL     MEM0B
0230 EF0000      R      LD      RESULT+2H,TMP2
0233 A01E02      R      LD      RESULT,TMP0
0236 A01C00      R      ;      STATEMENT 137
0239 ADF000      R      LDBZE    ERRORCODES,#0F0H
                        ;      STATEMENT 138
023C ENDTTESTS:  R      LD      TMP2,RESULT+2H
023C A0021E      R      LD      TMP0,RESULT
023F A0001C      E      POP      ?FRAME01
0242 CC00        E      LD      TMP6,[SP]
0244 A21822      ADD     SP,#16H
0247 65160018    BR      [TMP6]
024B E322        ;      STATEMENT 139
                        ;      STATEMENT 140
                        END

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 024DH    589D
CONSTANT AREA SIZE   = 0000H    0D
DATA AREA SIZE       = 0000H    0D
STATIC REGS AREA SIZE = 0004H    4D
OVERLAYABLE REGS AREA SIZE = 0000H  0D
MAXIMUM STACK SIZE   = 000AH   10D
183 LINES READ

```

PL/M-96 COMPILATION COMPLETE. 0 WARNINGS, 0 ERRORS

SPARC-PP COMPILE/COMPILE 0 HYBRIDGE 8 28002

183 FILES REVD
MCS-96 MACRO ASSEMBLER SELECTED TESTS_ASM96

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F5:D96FST.A96

OBJECT FILE: :F5:D96FST.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOGEN DEBUG

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			1	;
			2	*****
			3	Selected Tests_ASM96 MODULE STACKSIZE(20)
			4	***** 0031
			5	;
			6	; In order to run this module, the STACK must be ALL external, and the
			7	; data ram partitioned for memory test must not include ANY of the STACK
			8	;
			9	; To call this module
			10	;
			11	PUSH #<RAM segment1 start address>
			12	PUSH #<RAM segment1 ending address>
			13	PUSH #<RAM segment2 start address>
			14	PUSH #<RAM segment2 ending address>
			15	PUSH #<random seed>
			16	PUSH #<number of cycles desired for random test>
			17	PUSH #<address of the last byte of rom>
			18	PUSH #<an argument for mul/div tests>
			19	PUSH #<a second argument for mul/div tests>
			20	PUSH #<a bit pattern for memory tests>
			21	CALL D96FST
			22	;
			23	;
			24	*****
0000			25	rseg
			26	extrn sp,ereg1,ereg2
			27	;
			28	;
			29	;
			30	;
0000			31	cseg
			32	extrn sys01,sys02,sys03,alu01,alu02,alu03,alu04,alu05
			33	extrn mem01,mem02,mem03,mem04,mem05,mem06,mem07,mem08
			34	extrn mem09,mem0a,mem0b,mem0c,mem0d
			35	;
			36	PUBLIC D96FST
			37	\$reject

7-80

D96FST

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			38	
			39	\$include (:f3:dstmac.inc)
			40	*****
			41	;DST Macros INCLUDE FILE ;*****
			42	*****
			43	
			44	rseg
			45	
			46	macro_temp: DSB 1
			47	extrn zero,sp_stat
			48	
			49	cseg
			50	ti equ 5
			51	ri equ 6
			52	
			53	
			54	*****
			55	RESET_WATCHDOG MACRO ;macro to reset the watchdog
			56	LDB 0ah,#1eh
			57	LDB 0ah,#0elh
			58	ENDM
			59	*****
			60	
			61	*****
			62	
			63	SPSTATUS MACRO v1 ;macro to read sp_stat to
			64	LOCAL Sp_read ;work around the ri/ti bug
			65	Sp_read: ;on the 8x9x-90.
			66	LDB macro_temp,sp_stat
			67	ORB v1,macro_temp
			68	ANDB macro_temp,#01100000B
			69	JNE Sp_read
			70	ENDM
			71	*****
			72	
			73	
			74	*****
			75	
			76	SPWAIT MACRO v2 ;macro to wait for ri/ti set
			77	;and avoid 8x9x-90 bug.
			78	;NOTE!! this macro won't work
			79	;with a full duplex line.
			80	JBC sp_stat,v2,\$
			81	LDB zero,sp_stat
			82	ENDM
			83	*****
			84	
			85	
			86	
			87	*****
			88	BR_ON_ERR MACRO Label ;macro to test high byte of
			89	;EREGL and branch away if
			90	CMPB eregl+1h,zero ;the byte is non-zero (which
			91	BNE Label ;means there was an error).
			92	ENDM
			93	*****
			94	

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		96	
	0000	97	D96FST:
		98	
	0000 EF0000	99	CALL sys02.A96 ;CALL the Power Up Test
		100	
		101	BR_ON_ERR Error_Found
		105	
	000A EF0000	106	CALL alu01 ;CALL the Add/Subtract test
		107	
		108	BR_ON_ERR Error_Found
		112	
	0014 EF0000	113	CALL alu02 ;CALL the MULUB test
		114	
		115	BR_ON_ERR Error_Found
		119	
	001E EF0000	120	CALL alu03 ;CALL the Multiply/Divide Table
		121	; driven test
		122	BR_ON_ERR Error_Found
		126	
	0028 CB000C	127	PUSH 0ch[sp] ;PUSH a seed and test length
	002B CB000C	128	PUSH 0ch[sp] ; for the random number based
	002E EF0000	129	CALL alu04 ; Multiply/Divide Random test
		130	
		131	BR_ON_ERR Error_Found
		135	
	0038 CB0006	136	PUSH 06h[sp] ;PUSH an argument
	003B CB0006	137	PUSH 06h[sp] ;PUSH another argument
	003E EF0000	138	CALL alu05 ;CALL the Multiply/Divide Core test
		139	
		140	BR_ON_ERR Error_Found
		144	
	0048 EF0000	145	CALL mem01 ;CALL the Complementary Address test
		146	; for internal registers
		147	BR_ON_ERR Error_Found
		151	
	0052 EF0000	152	CALL mem03 ;CALL the Galloping 1s/0s test
		153	; for internal registers
		154	BR_ON_ERR Error_Found
		158	
	005C EF0000	159	CALL mem05 ;CALL the Chekerboard Pattern test
		160	; for internal registers
		161	BR_ON_ERR Error_Found
		165	
	0066 CB0014	166	PUSH 14h[sp] ;PUSH a start and end address
	0069 CB0014	167	PUSH 14h[sp] ; for a region of RAM to conduct
	006C EF0000	168	CALL mem06 ; the Complementary Address test for
		169	; external RAM
		170	BR_ON_ERR Error_Found
		174	
	0074 CB0010	175	PUSH 10h[sp] ;PUSH a start and end address
	0077 CB0010	176	PUSH 10h[sp] ; for another region of RAM to conduct
	007A EF0000	177	CALL mem06 ; the Complementary Address test for
		178	; external RAM
		179	BR_ON_ERR Error_Found
		183	
	0082 CB0014	184	PUSH 14h[sp] ;PUSH a start and end address
	0085 CB0014	185	PUSH 14h[sp] ; for a region of RAM, and PUSH

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT	
	0088	CB0006	E 186	PUSH 06h[sp]	
	008B	EF0000	E 187	CALL mem0d	; a bit pattern to use in the
			188		; Checkerboard Pattern test for
			189	BR_ON_ERR Error_Found	; external RAM
	0093	CB0010	E 194	PUSH 10h[sp]	
	0096	CB0010	E 195	PUSH 10h[sp]	; PUSH a start and end address for
	0099	CB0006	E 196	PUSH 06h[sp]	; another region of RAM, and PUSH
	009C	EF0000	E 197	CALL mem0d	; a bit pattern to use in the
			198		; Checkerboard Pattern test for
			199	BR_ON_ERR Error_Found	; external RAM
	00A4	CB0014	E 204	PUSH 14h[sp]	
	00A7	CB0014	E 205	PUSH 14h[sp]	; PUSH a start and end address
	00AA	EF0000	E 206	CALL sys03	; for a region of RAM to conduct
			207		; the Program Counter test
			208	BR_ON_ERR Error_Found	
	00B2	CB0010	E 213	PUSH 10h[sp]	
	00B5	CB0010	E 214	PUSH 10h[sp]	; PUSH a start and end address
	00B8	EF0000	E 215	CALL sys03	; for another region of RAM to conduct
			216		; the Program Counter test
			217	BR_ON_ERR Error_Found	
	00C0	C98020	E 222	PUSH #2080h	
	00C3	CB000A	E 223	PUSH 0ah[sp]	; PUSH the code starting address
	00C6	EF0000	E 224	CALL mem0b	; PUSH the code ending address
			225		; CALL the checksum routine
	00C9	A1310000	E 226	LD eregl,#0031h	; ALL DIAG96 TESTS PASSED
			227		; load appropriate error code
	00CD	CF0014	E 229	POP 14h[sp]	
	00D0	65120000	E 230	ADD sp,#12h	; clean off the stack
	00D4	F0	E 231	RET	
			232		; return to the calling program
	00D5		E 233	Error_Found:	
			234		
	00D5	CF0014	E 235	POP 14h[sp]	; clean off the stack
	00D8	65120000	E 236	ADD sp,#12h	
	00DC	F0	E 237	RET	; return to the calling program
			238		
	00DD		E 239	;*****	
			240	end	

SYMBOL TABLE LISTING

N A M E	VALUE	ATTRIBUTES
ALU01	----	CODE EXTERNAL
ALU02	----	CODE EXTERNAL
ALU03	----	CODE EXTERNAL
ALU04	----	CODE EXTERNAL
ALU05	----	CODE EXTERNAL
BR ON ERR	----	MACRO
D96FST	0000H	CODE REL PUBLIC ENTRY
EREG1	----	REG EXTERNAL
EREG2	----	REG EXTERNAL
ERROR FOUND	00D5H	CODE REL ENTRY
MACRO TEMP	0000H	REG REL BYTE
MEM01	----	CODE EXTERNAL
MEM02	----	CODE EXTERNAL
MEM03	----	CODE EXTERNAL
MEM04	----	CODE EXTERNAL
MEM05	----	CODE EXTERNAL
MEM06	----	CODE EXTERNAL
MEM07	----	CODE EXTERNAL
MEM08	----	CODE EXTERNAL
MEM09	----	CODE EXTERNAL
MEM0A	----	CODE EXTERNAL
MEM0B	----	CODE EXTERNAL
MEM0C	----	CODE EXTERNAL
MEM0D	----	CODE EXTERNAL
RESET WATCHDOG	----	MACRO
RI	0006H	NULL ABS
SELECTED TESTS_ASM96	----	MODULE STACKSIZE(20)
SP	----	REG EXTERNAL
SP STAT	----	REG EXTERNAL
SPSTATUS	----	MACRO
SPWAIT	----	MACRO
SYS01	----	CODE EXTERNAL
SYS02	----	CODE EXTERNAL
SYS03	----	CODE EXTERNAL
TI	0005H	NULL ABS
ZERO	----	REG EXTERNAL

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

MCS-96 MACRO ASSEMBLER DSTUSR

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:DSTUSR.A96

OBJECT FILE: :F2:DSTUSR.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: GEN DEBUG

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			1	;*****
			2	DSTUSR MODULE main,stacksize(2)
			3	;*****
			4	
			5	
			6	oseg at 400h
			7	User_Registers: DSB lch
			8	temp set User_Registers:WORD
			9	
			10	rseg
			11	EXTRN sp,zero,timer1,ereglord
			12	
			13	
			14	dseg at 100h ;to ensure that the STACK does not get
			15	
			16	DSEG1: DSB 700h ; located in an area of RAM that will be
			17	
			18	dseg at 4200h ; memory tested, reserve those regions
			19	
			20	DSEG2: DSB 1e00h ; as data segments.
			21	
			22	
			23	
			24	cseg at 2080h
			25	
			26	extrn alu04,alu01,alu02,mem06,mem0a,error_proc,alu05
			27	EXTRN DSTISR
			28	
			29	LD temp,#0ffh
			30	DJNZ temp,\$; wait for sbe96 NMIs to stop
			31	
			32	LD sp,#STACK
			33	
			34	PUSH #100h ;RAM segment1 start address
			35	PUSH #7ffh ;RAM segment1 end address
			36	PUSH #4200h ;RAM segment2 start address
			37	PUSH #5fffh ;RAM segment2 end address
			38	PUSH #47efh ;random seed
			39	PUSH #1000h ;random test length
			40	PUSH #3fffh ;top of code address
			41	PUSH #9d42h ;an argument for mul/div test
			42	PUSH #778ch ;another argument for mul/div test
			43	PUSH #5a5ah ;bit pattern for memory test
			44	CALL DSTISR ;CALL the Dynamic Stability Test
			45	
			46	
			47	Main_Task: ;
			48	
			49	push #8080h ;
			50	push #8000h ; use the multiply/divide core
			51	call alu05 ; test on the arguments

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT	
	20B5	980001	E 52	cmpb eregl+1,zero	; #8080h and #8000h in all
	20B8	DF022074	53	bne error_found	; combinations
				IJE \$+4	
				ISJMP error_found	; combinations
	20BC	C90080	54		
	20BF	C98080	55	push #8000h	
	20C2	EF0000	56	push #8080h	
	20C5	980001	E 57	call alu05	
	20C8	DF022064	E 58	cmpb eregl+1,zero	
			59	bne error_found	
				IJE \$+4	
				ISJMP error_found	
	20CC	C90080	60		
	20CF	C90080	61	push #8000h	
	20D2	EF0000	62	push #8080h	
	20D5	980001	E 63	call alu05	
	20D8	D756	E 64	cmpb eregl+1,zero	
			65	bne error_found	
	20DA	C98080	66		
	20DD	C90080	67	push #8080h	
	20E0	EF0000	68	push #8000h	
	20E3	980001	E 69	call alu05	
	20E6	D748	E 70	cmpb eregl+1,zero	
			71	bne error_found	
	20E8	C90001	72		
	20EB	C9FF07	73	push #100h	; perform a galloping ls/0s test
	20EE	EF0000	74	push #7ffh	; on a small section of RAM
	20F1	980001	E 75	Call mem0a	
	20F4	D73A	E 76	cmpb eregl+1,zero	
			77	bne error_found	
	20F6	C800	78		
	20F8	C90020	E 79	push timer1	; send a timer1 based seed to the
	20FB	EF0000	80	push #2000h	; random number based multiply/divide
	20FE	980001	E 81	call alu04	; test and let it run for a string
	2101	D72D	E 82	cmpb eregl+1h,zero	; of 2000h argument pairs
			83	bne error_found	
	2103	EF0000	84		
	2106	980001	E 85	call alu01	; perform the add/subtract test
	2109	D725	E 86	cmpb eregl+1h,zero	
			87	bne error_found	
	210B	C90042	88		
	210E	C9FF5F	89	push #4200h	; perform a Complementary address test
	2111	EF0000	90	push #5fffh	; on a large section of RAM
	2114	980001	E 91	Call mem06	
	2117	D717	E 92	cmpb eregl+1,zero	
			93	bne error_found	
	2119	EF0000	94		
	211C	980001	E 95	call alu02	; perform the MULUB test
	211F	D70F	E 96	cmpb eregl+1h,zero	
			97	bne error_found	
	2121	C800	98		
	2123	C90020	99	push timer1	; send another timer1 based seed to
	2126	EF0000	100	push #2000h	; the random number based multiply/
	2129	980001	E 101	call alu04	; divide test
	212C	D702	E 102	cmpb eregl+1h,zero	
			103	bne error_found	
			104		

MCS-96 MACRO ASSEMBLER DSTUSR

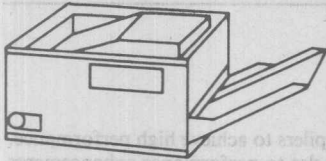
ERR LOC	OBJECT	LINE	SOURCE STATEMENT	
212E	277C	105	BR Main_task	; start the main_task tests over
		106		
2130		107	Error_found:	
2130	FA	108	di	; if an error is found, disable
2131	EF0000	109	CALL Error_Proc	; interrupts and call the error
		110		; procedure in the DST96.LIB.
		111		; the test that found an error will
		112		; have placed the appropriate
		113		; error codes in locations EREG1 and
		114		; EREG2 for output through Error_Proc
2134	27FE	115	BR \$	
		116	;*****	
		117		
2136		118	end	

M S-96 MACRO ASSEMBLER DSTUSR

SYMBOL TABLE LISTING

N A M E	VALUE	ATTRIBUTES
ALU01	----	CODE EXTERNAL
ALU02	----	CODE EXTERNAL
ALU04	----	CODE EXTERNAL
ALU05	----	CODE EXTERNAL
DSEG1	0100H	DATA ABS BYTE
DSEG2	4200H	DATA ABS BYTE
DSTISR	----	CODE EXTERNAL
DSTUSR	----	MODULE MAIN STACKSIZE(2)
EREG1	----	REG EXTERNAL
ERROR_FOUND	2130H	CODE ABS ENTRY
ERROR_PROC	----	CODE EXTERNAL
MAIN_TASK	20ACH	CODE ABS ENTRY
MEM06	----	CODE EXTERNAL
MEM0A	----	CODE EXTERNAL
SP	----	REG EXTERNAL
TEMP	0040H	OVERLAY ABS WORD
TIMER1	----	REG EXTERNAL
USER_REGISTERS	0040H	OVERLAY ABS BYTE
ZERO	----	REG EXTERNAL

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.



Intel's 80960: An Architecture Optimized for Embedded Control

Intel's internally developed 80960 architecture allows embedded system designers to take advantage of silicon technology advancements without architectural limits. The 80960, developed from scratch for embedded control applications, eliminates architectural obstacles to state-of-the-art implementation techniques that allow parallel and out-of-order instruction execution.

In introducing the 80960 architecture, I distinguish between the architecture and the microarchitecture of an implementation. A microarchitecture is an actual implementation of the architecture's instruction set and programming resources. Different microarchitectures may have different pipeline construction, internal bus widths, register set porting, cache parameterization (two-way, four-way, and so on), and degrees of parallelism, or may not execute instructions out of order. The architecture is specified in such a way that wide latitude is available for future microarchitectural advancements. In this way both very high performance and highly integrated controllers can be built using the 80960 architecture.

Principally, the 80960 architecture allows, for at least the next decade, silicon technology and microarchitectural advances to translate directly into increased performance without architectural limitations. While the common performance target of typical RISC architectures is an execution speed of one instruction per processor clock cycle, the 80960 architecture targets the execution of multiple instructions per clock cycle (fractional clock cycles per instruction). By defining an architecture that supports parallel instruction execution and out-of-order instruction execution, we do not constrain performance advances to the system clock cycle.

Additionally, the 80960 has been optimized for the wide range of applications that are unencumbered by a need to be compatible with an existing embedded control architecture. These applications are often very cost sensitive, require a different mix of instructions than reprogrammable applications, have demanding interrupt response requirements, and use real-time executives rather than full-blown operating systems. With these factors in mind, we developed the 80960 while avoiding architectural constructs that would restrict an implementation's capability to execute multiple instructions in one clock cycle.

**Executing
instructions in
one clock cycle is
not fast enough
for this standard-
core architecture.
Its parallelism
and out-of-order
execution promise
fractional clock
rates in future
implementations.**

David P. Ryan
Intel Corporation

80960 architecture

The architecture also allows application-specific extensions such as:

- instruction set extensions (floating-point operations),
- special registers,
- larger caches,
- multiple caches,
- on-chip program and data memory,
- a memory management and protection unit,
- fault-tolerance support,
- multiprocessing support, and
- real-time peripherals (DMA, analog/digital, serial ports).

The 80960's instruction set has also been optimized for embedded control applications. It offers Boolean operations more powerful than those of the 8051 family. Single instructions access frequently executed functions for increased code density and performance. They include CALL, RET, Compare__and__Branch, Conditional__Compare, Compare__and__Increment or Decrement, and Bit Field Extract.

A priority interrupt structure simplifies the management of real-time events, and, along with a user/supervisor model, supports fast, safe, real-time kernel operation. A generalized fault-handling mechanism simplifies the task of detecting errant arithmetic calculations or other conditions that typically require a significant amount of user code runtime support. The 80960 does not require sophisticated, optimizing high-

level-language compilers to achieve high performance. However, no obstacles to performance enhancements via a good optimizing compiler exist.

Since products based on the 80960 have high performance levels, even without code optimization, many users will attain their required system performance with 80960 products without having to understand the parallelism of the implementation they are using.

Architecture overview

The 80960 can best be described as a register-rich, load/store architecture with an instruction set designed to let implementations exploit pipelining and parallel execution strategies. Direct embedded control support includes:

- an optimized instruction set,
- a flexible interrupt structure,
- a user-supervisor model,
- a powerful fault-handling structure, and
- multiprocessing hooks and debug support.

The architecture extends easily.

Figure 1 shows a logical block diagram of the architecture's programming resources. The 32-bit memory space is flat. Data moves between memory and registers via load and store instructions. Processing elements surrounding the registers manipulate parallel data; they receive their instructions from the

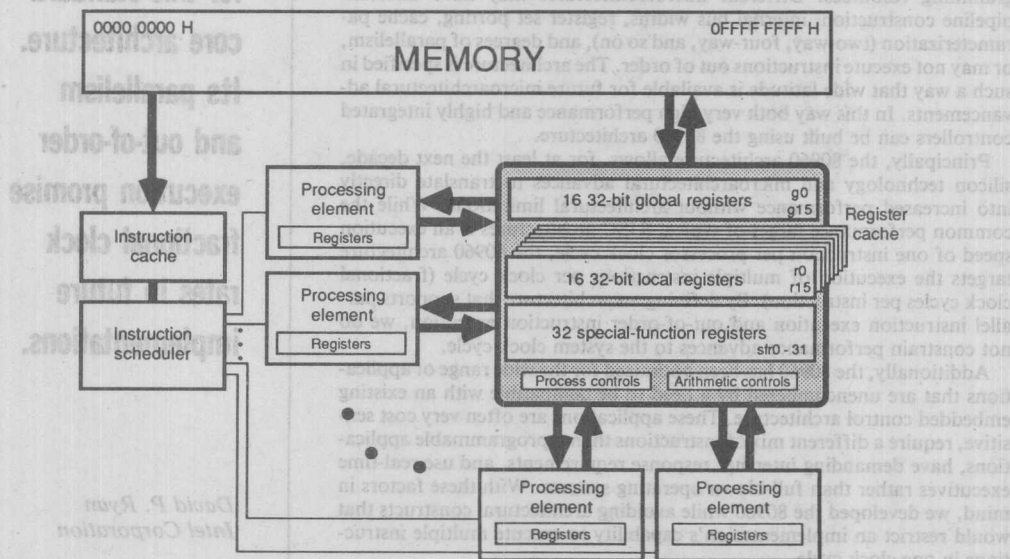


Figure 1. Block diagram of the 80960 architecture.

instruction sequencer.

The instruction sequencer reads multiple instructions simultaneously from an instruction cache and presents the instructions in parallel to the processing elements as appropriate. When the instruction stream or an asynchronous event requires a context switch, the local variables from the suspended procedure move to the register cache. Memory accesses to save previously suspended register sets occur only when the register cache is filled. The implementation determines the number of architecturally transparent register sets that can be cached.

We expect different implementations of the processing elements in an 80960-based controller—optimized for specific applications—will evolve. We defined a standard core architecture to maintain binary-compatible instruction sets across all implementations for leveraging compiler and real-time kernel development. Subsequent references to the 80960, without an alpha suffix, refer to the architecture. References to an 80960.XY pertain to actual implementations of the core architecture, which may contain architectural extensions and/or on-chip peripherals. (See the accompanying box for a discussion of three implementations.)

Implementations of the 80960

As an example of an actual implementation of the core architecture, consider the first 80960 implementation, the 80960KB controller. The KB provides architectural extensions such as floating-point operations (Figure A). Its on-chip floating-point unit implements the IEEE floating-point standard, including transcendental support. Floating-point performance exceeds 4 million Whetstone operations per second at 20 MHz. The 80960KB integrates a 512-byte instruction cache and an interrupt controller on chip.

Another member of the family, the 80960KA controller, fits the KB socket but lacks the on-chip floating-point unit. The 80960MC controller, a military-qualified version similar to the KB, adds Ada tasking support and a memory management and protection unit.

The 80960KA, KB, and MC microarchitecture sustains execution of up to one instruction per clock cycle at 20 MHz (20 native MIPS). It performs a variety of benchmark programs seven to 10 times faster than a VAX 11/780 (7 to 10 VAX MIPS).

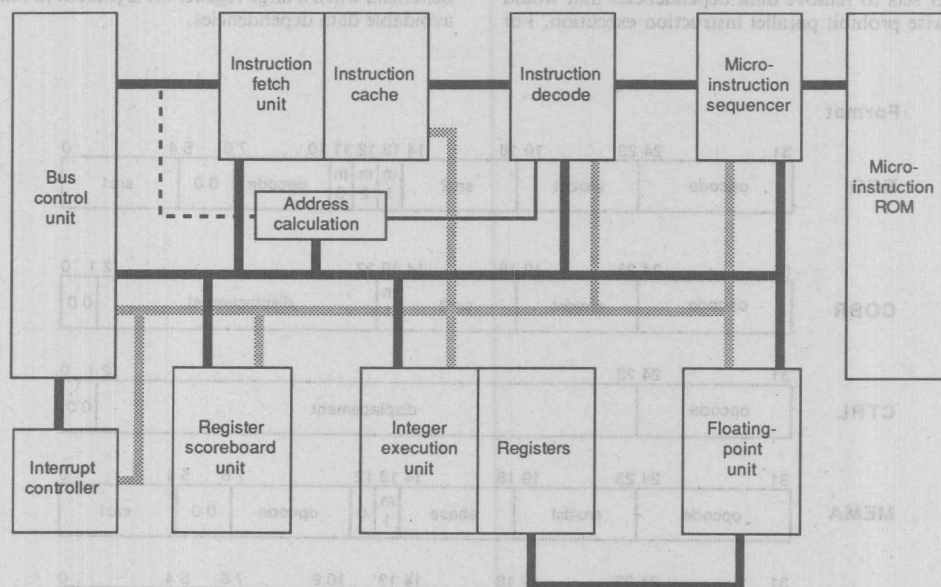


Figure A. Block diagram of the 80960KB controller.

80960 architecture

Register model

The user directly accesses thirty-two 32-bit general-purpose registers and 32 special-function registers (SFRs). (Refer to Figure 1.) Of the 32 general registers, 16 are global registers and 16 are local registers. Data in the 16 global registers remain visible and unchanged when crossing procedure boundaries, characteristics exhibited by "normal" registers in other architectures. The CALL instruction caches the local registers and the RET instruction restores them. The SFRs provide a real-time register interface to on-chip peripherals. The SFRs, the contents of which are not defined by the architecture, control implementation-specific hardware.

When a procedure call occurs, data in the local registers automatically move to a register cache. Thus, the called procedure is not required to explicitly save the local registers. When the called procedure executes a return instruction, the data in the local registers prior to the call are restored. The call/return sequence does not affect the global registers, which can be used to pass parameters and results between procedures.

A large, general-purpose register set greatly reduces the number of memory requests required to perform a task. Various optimization techniques can use large-register sets to remove data dependencies that would otherwise prohibit parallel instruction execution. For

example, a hypothetical implementation of the architecture could provide parallel execution of two ADD instructions. Having many general-purpose registers with which to work simplifies code optimization so that neither ADD instruction references a source that was the destination of the other ADD instruction. Under such circumstances, two ADD instructions per cycle could be sustained.

Note that such program optimizations are not required. Any program using the architecture's core instruction set and not referencing the SFR address space or external I/O, whether optimized or not, will function correctly on any implementation without modification. Even if the optimization rules are radically different between implementations, code that is optimized for one implementation will run correctly on another implementation without modification or recompilation.

The architecture also guarantees that data dependencies will be correctly handled without programmer intervention. For example, execution of an arithmetic instruction will be delayed if it uses a register that is waiting for data to be loaded from memory. However, other instructions that do not use the register in question could be executed immediately with all data dependencies automatically maintained. This capability is only beneficial when a large register set is present to remove avoidable data dependencies.

Format

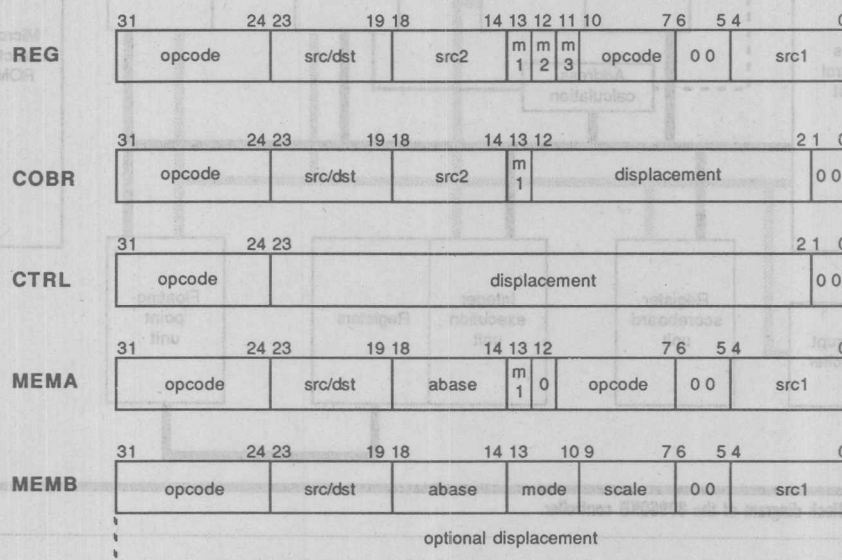


Figure 2. Opcode encodings.

Core instruction set

The 80960 instruction set is similar in design to engines in reduced instruction set computers. Because the 80960 was designed to avoid performance bottlenecks resulting from instruction decoding times, all opcodes are 32 bits (one word) in length and must be aligned on word boundaries. The only two-word (64-bit) instruction format loads 32-bit immediate constants and assists effective address calculations. Generally, load, store, and control instructions access memory. All other instructions access only registers.

Thirty-two-bit opcodes provide tremendous flexibility in instruction encoding. However, performance penalties associated with code size can occur when often-used complex instruction sequences are not available in a one-word format. Large code size not only increases system cost due to larger memory requirements but also results in penalties in cache utilization and bus-bandwidth requirements.

The 80960 includes one-word multifunction instructions to avoid such code density problems that increase memory requirements and to allow complex functions to be parallelized. For example, the CALL and RET instructions provide one-word encodings for sequences that otherwise require several instructions. Implementations of the architecture could perform the CALL/RET operations in parallel with other instruction execution. Or, the processor could execute from an on-chip ROM a similar sequence of simple 80960 instructions that the user would have to write if CALL/RET instructions were not in the architecture. Executing the code from permanent on-chip storage results in higher performance than does requiring the instructions to be fetched and cached every time they are used. In addition to ensuring higher performance, or possibly parallel performance, implementations of such functions as ROMed assembly code sequences—triggered by a one-word opcode—are more silicon efficient than are increases of on-chip cache sizes to deal with poor code density. For example, a ROM cell is typically one fourth the size of a RAM cell.

The availability of complex instructions in the architecture does not prohibit the programmer from bypassing them if simpler functions are desired. For example, a BAL (Branch and Link) instruction can be used to call a procedure that does not require a new set of local registers. The BAL instruction saves the next instruction pointer in a register and branches to the specified destination. When the procedure is ready to return, it executes an indirect branch to the return instruction pointer that was saved. It is likely that BAL will always be faster than CALL since no local registers are being saved. The programmer can use whichever method best suits the circumstances.

Figure 2 shows the 80960 instruction encodings. Most instructions appear in the REG format, which specifies an opcode and three registers/literals (one of 32 registers, or a constant value in the range 0 to 31).

The COBR format specifies a set of compare and branch instructions. The CRTL format covers branch and call instructions. The MEM formats support load and store instructions.

Much of the instruction set is what one would expect to encounter in all processors (ADD, MUL, SHIFT, BRANCH); however, some instructions deserve special mention.

- The register-register move instructions transfer one, two, three, or four register values. The same is true for the load and store instructions (for example, LDQ loads four words into four registers).
- In addition to the normal shift instructions, the SHRDI instruction provides an adjustment to the result so the instruction can be used to divide a value by a power of two. (Normal right-shift instructions do not divide correctly when the value is negative and odd.)
- All logical operations of two operands are provided (AND, NOTAND, ANDNOT, NOR).
- An extensive set of bit instructions exists (SET BIT, CLEAR BIT, NOT BIT, SCAN a register for the first 0, or 1, BRANCH on bits set or clear), as well as instructions for accessing bit fields (MODIFY, EXTRACT).
- Single-instruction COMPARE_AND_BRANCH encodings optimize code density for these frequently executed operations.
- Conditional compare (CONCOMP) instructions speed bounds checking.

Table 1 on the next page summarizes the 80960 core architecture instruction set.

The architecture directly supports integer (signed) and ordinal (unsigned) data types. Bits, bit fields, bytes, short words, words, and double words can be manipulated in registers and transferred to and from memory. Triple words and quad words can also move between the registers and memory.

Register operations

Most 80960 instructions operate on registers. The architecture provides arithmetic, logical, bit and fit field, data movement, and comparison operations. To take full advantage of the large register set, three-operand instructions specify any register as one or both sources and/or the destination of an operation.

Arithmetic and logical. The architecture supports both standard and extended arithmetic operations. Add, subtract, multiply, and divide operations are available on 32-bit integers and ordinals. Extended multiply operates on two 32-bit ordinals and generates a 64-bit result. Extended divide divides a 64-bit ordinal by a 32-bit ordinal, producing a 32-bit quotient and 32-bit remainder. Addition and subtraction with carry allow 32-bit ordinals to provide extended precision adds and subtracts.

80960 architecture

Table 1.
80960 instruction summary.

REGISTER OPERATIONS	
ARITHMETICS	
add[i o]	Add
addc	Add with Carry
sub[i o]	Subtract
subc	Subtract with Borrow
mul[i o]	Multiply
emul	Extended Multiply
div[i o]	Divide
ediv	Extended Divide
rem[i o]	Remainder
modi	Modulo Integer
sh[lo ro li ri di]	Shift
MOVEMENT	
mov[t q]	Move registers to registers
lod	Load Address
COMPARISON	
cmp[i o]	Compare
cmpdec[i o]	Compare and Decrement
cmpinc[i o]	Compare and Increment
concomp[i o]	Conditional Compare
test[*]	Test for condition
scanbyte	Scan for matching byte
LOGICAL	
and	dst := src1 & src2
andnot	dst := src2 & (~src1)
notand	dst := (~src2) & src1
nand	dst := ~(src2 & src1)
or	dst := src1 src2
nor	dst := ~(src2 src1)
ornot	dst := src2 (~src1)
notor	dst := (~src2) src1
xnor	dst := (src2 src1) & ~(src2 & src1)
xor	dst := ~(src2 src1) (src2 & src1)
not	dst := ~src
rotate	Rotate Bits
BIT AND BIT FIELD	
setbit	Set a Bit
clrbit	Clear a Bit
notbit	Toggle (invert) a Bit
chkbitt	Check a Bit and set condition code
alterbit	Change a Bit according to an operand
scanbit	Search src for most significant set bit
spanbit	Search src for most significant cleared bit
extract	Extract specified bit pattern from a word
modify	Modify selected bits in dst with src
CONTROL OPERATIONS	
BRANCH	
b	Branch (± 2 MByte relative offset)
bx	Branch Extended (32-Bit Indirect Branch)
bal[x]	Branch and Link ("RISC Branch")
b[[*]]	Branch on Condition
cmpbit[[*]]	Compare Integer and Branch on Condition
cmpob[[*]]	Compare Ordinal and Branch on Condition
FAULT	
fault[[*]]	Fault on Condition
syncf	Synchronize Faults
PROCEDURE	
call	Procedure Call (± 2 MByte relative offset)
callx	Call Extended (32-Bit Indirect Call)
calls	System Procedure Call
ret	Return
ENVIRONMENT	
modpc	Modify Process Controls
modac	Modify Arithmetic Controls
modtc	Modify Trace Controls
flushregs	Flush Local Register Cache to Memory
DEBUG	
mark	Conditionally generate Trace Fault
fmark	Unconditionally generate Trace Fault
MEMORY OPERATIONS	
LOAD/STORE	
ld[b s l t q]	Load
st[b s l t q]	Store
READ/MODIFY/WRITE	
atadd	Atomic Add (Locked RMW Cycles)
atmod	Atomic Modify (Locked RMW Cycles)

Arithmetic shift operations support 32-bit ordinals. Logical shift instructions operate on 32-bit integers, and a 32-bit register value can also be rotated. In addition, all possible two-operand, bitwise Boolean operations exist: AND, NOTAND, ANDNOT, XOR, OR NOR, XNOR, NOT, NOTOR, ORNOT, and NAND.

Bit and bit field. Bit operations allow bits in the registers to be set, cleared, toggled, and moved to or from the condition codes. SCAN and SPAN operations provide ways to find the most significant set or cleared bit in a register.

The 80960 contains two bit field instructions, EXTRACT and MODIFY. The EXTRACT instruction shifts a bit field in a register to the right and fills in the bits to the left of the bit field with zeros. The MODIFY instruction moves a specified bit field from one register to another when no adjustment change in bit position is required.

Data movement. A set of data movement (MOV) instructions allows register values to be copied to other registers. The MOV instructions can move from one to four registers at once. The load and store operations described later move data to and from memory.

Comparison. These instructions compare operands and set the resulting condition codes in the arithmetic controls register (Figure 3). The 80960's condition codes listed in Table 2 provide the arithmetic flag function of other architectures, in a way that allows maximum parallel execution.

In general, only compare instructions set the 80960's condition codes and conditional instructions use them. To perform an ADD followed by a conditional branch when the result is zero, a Compare and Branch instruction must be executed after the ADD because arithmetic instructions do not alter the condition codes.

Table 2.
Condition code encodings.

Condition code	Condition
000	Unordered *
001	Greater Than
010	Equal (True)
011	Greater Than or Equal
100	Less Than
101	Not Equal (False)
110	Less Than or Equal
111	Ordered

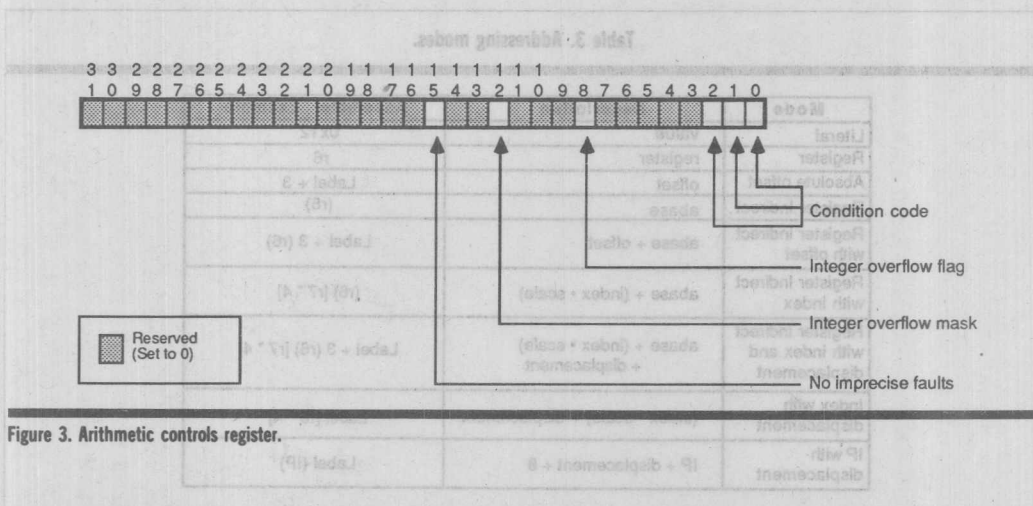
* Used with floating-point data types.

Although not generally noticed in a sequential execution environment, a parallel environment mandates the decoupling of the condition codes from the instruction set. The 80960 allows multiple instructions to execute simultaneously, thus giving ambiguous meaning to a set of condition codes that are altered by multiple arithmetic instructions in the same clock cycle. The 80960 approach separates condition checking and decision making from all other instructions to provide flexibility in reordering instructions for parallel execution.

The 80960 compare instructions compare both integers and ordinals. A subset of the compare instructions increments or decrements an operand after the comparison.

Memory operations

The load/store nature of the architecture decouples memory references from instruction execution. Register set and memory instructions can be executed simultaneously. Since the load data may take some time to



80960 architecture

arrive at the CPU, the load requests can be advanced in the instruction stream to overlap memory access time with other data-independent CPU operations.

Load/store. The load and store instructions copy bytes, short words, words, or multiple words to or from memory and registers. When a load integer is specified for an 8-bit or 16-bit quantity, the CPU extends the data's sign to fill 32 bits before writing the destination register. When a load ordinal is specified for an 8-bit or 16-bit quantity, the CPU attaches leading zeros to the data to fill 32 bits before writing the destination register. The store instructions allow the destination data width to be a byte, short word, word, or multiple words. When a store byte, or short word is performed, the CPU automatically formats the data being written according to the data type (integer or ordinal).

Addressing modes. The architecture supports 11 addressing modes for memory operations, as summarized in Table 3. The addressing modes selected for support provide a broad range of most-often-used simple modes. We chose a rich set of addressing modes to allow optimization for code density as well as speed.

Literals are immediate 5-bit numbers that can range from 0 to 31. Literals may be used as operands in any register operation.

The **Register** address mode is used when an operand specifies a register number (g0, r5).

The **Absolute Offset** address mode specifies the absolute address of the target as an offset from the current instruction pointer. The offset is encoded in the memory instruction opcode. If the offset is outside the range of 0 to 2048, the assembler generates a two-word

instruction in which the second word is a 32-bit displacement.

Register Indirect addressing allows the address of the target to be specified by the contents of a register. An immediate offset or displacement can be added to the register to form the effective address. An index (scaled by 2, 4, 8, or 16) may also be added.

Memory operations can also specify target addresses relative to the instruction pointer, a capability useful in creating relocatable data and code.

Atomic memory operations. Two atomic memory operations support multiprocessing environments with more than one processor accessing the same memory, atomic add (ATADD) and atomic modify (ATMOD). The ATADD instruction causes an operand to be added to the value in the specified memory location. The ATMOD causes bits in the specified memory location to be modified under control of a mask. These instructions perform their memory-to-memory, read-modify-write operations with a locked bus to prevent data corruption.

Control operations

Control operations include those instructions that could result in the redirection of program flow. CALL, RET, BRANCH, and COMPARE AND BRANCH instructions fall into this category.

Procedure calls. The CALL instruction causes the local registers to be preserved and redirects program flow to a point indicated by an offset encoded in the instruction. The Call Extended (CALLX) instruction dif-

Table 3. Addressing modes.

Mode	Description	Assembler Example
Literal	value	0x12
Register	register	r6
Absolute offset	offset	Label + 3
Register Indirect	abase	(r6)
Register Indirect with offset	abase + offset	Label + 3 (r6)
Register Indirect with index	abase + (index * scale)	(r6) [r7 * 4]
Register Indirect with index and displacement	abase + (index * scale) + displacement	Label + 3 (r6) [r7 * 4]
Index with displacement	(index * scale) + displacement	Label [r6 * 4]
IP with displacement	IP + displacement + 8	Label (IP)

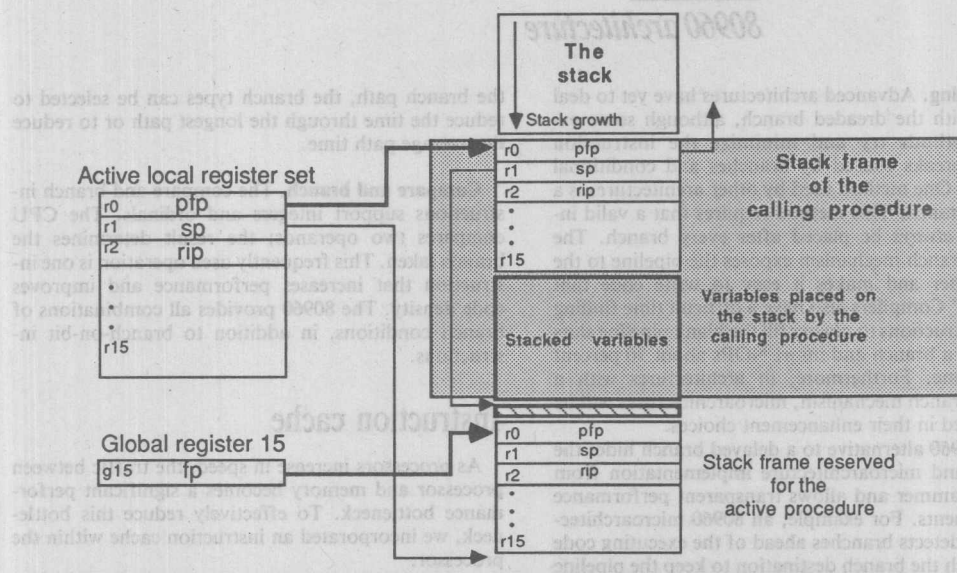


Figure 4. Procedure stack structure.

fers in that it allows a 32-bit value to provide the CALL destination. The destination can either be encoded in the instruction or specified by a register value (for example, indirect call). The Call System (CALLS) instruction gets its target address from a table of system procedure addresses explained later. The Return (RET) instruction returns control to the calling procedure and restores the local registers of the calling procedure.

The semantics of the CALL/RET allow an optimization known as register caching. A register cache keeps the context (local registers) of the most recently executing subroutines on chip so that CALL/RET instructions do not have to access memory to save or restore the local registers.

When a CALL instruction executes, the 80960 allocates a new set of 16 local registers from a pool of register sets for the called procedure. If the pool is depleted, a new register set is allocated by taking one associated with an earlier procedure and saving it in memory. A RET instruction causes the most recently cached local register set to be restored, freeing a register cache location.

The register cache contributes to performance in four ways:

- It significantly reduces the saving and restoring of registers that are usually performed when crossing subroutine boundaries.
- Since the local register sets are mapped into the stack frames, the linkage information that normally appears in stack frames (pointer to previous frame, saved instruction pointer) is contained in the local reg-

isters. Most call and return instructions execute without causing any references to off-chip memory.

- It allows compilers to map most or all of a procedure's local variables directly into registers.
- It provides a large number of registers (16 local and 16 global), which can be exploited by optimizing compilers.

The procedure stack (see Figure 4) reserves space for the cached registers of suspended procedures. When a register set must be flushed from the register cache to memory, it moves to the reserved stack frame space.

When a new procedure is entered, the 80960 allocates space for the procedure's register set as the only contents of its stack frame, although no memory accesses will occur unless the register cache is full. If the procedure desires more space on the stack for autovariables or parameter passing, it adjusts the stack pointer to reserve as much space as it needs. The procedure can then access this space using stack pointer relative addressing so long as the procedure is active. When the procedure returns, its stack is automatically reclaimed.

Branch and Link (BAL) performs a procedure call without saving the local registers. The 80960 saves the return instruction pointer in a global register and redirects program flow. To return from a routine that is invoked by a BAL, a BX (Branch Extended) is performed. BAL allows fast subroutine calls to leaf procedures without allocating (and possibly displacing) a new register set. Since a leaf procedure calls no other procedure, its registers can be allocated out of those remaining in the current set.

80960 architecture

Branching. Advanced architectures have yet to deal cleanly with the dreaded branch, although some existing methods try and minimize the instruction pipeline breaks caused by branches and conditional branches. One method used by other architectures is a delayed branch. This method requires that a valid instruction *always* be placed after every branch. The delayed branch mechanism exposes the pipeline to the programmer and makes it easy to write code that “breaks.” Compilers also have a difficult time finding useful instructions to *always* fill the blank pipeline slots following a branch and insert NOPs about 30 percent of the time. Furthermore, in architectures with a delayed branch mechanism, microarchitectures will be constrained in their enhancement choices.

The 80960 alternative to a delayed branch hides the pipeline and microarchitecture implementation from the programmer and allows transparent performance enhancements. For example, an 80960 microarchitecture that detects branches ahead of the executing code could fetch the branch destination to keep the pipeline full. In essence, “branch lookahead” allows branches to be executed in zero clock cycles.

Branches can be unconditional or conditional. The Branch and Branch Extended instructions perform unconditional redirection of program flow without linkage. Branch and Branch Extended differ in the width of the target address offset provided. The Branch instruction includes an encoded offset in the one-word instruction (MEMA format), whereas Branch Extended branches to the location pointed to by a register or an encoded 32-bit displacement (MEMB format).

The conditional branches use the condition codes in the arithmetic controls register to determine whether or not to take the branch. The 80960 provides all combinations of branch conditions.

Branch lookahead works well with unconditional branches but would be of marginal benefit on conditional branches since the branch target, or the instruction after the branch, cannot be executed until after evaluation of the branch condition. Pipeline breaks would, therefore, be inevitable even if the microarchitecture implemented some sort of hardware prediction mechanism. To reduce the effect of the conditional branch on performance, the 80960 defines two types of conditional branches, those that are usually taken and those that aren't usually taken. The implementation can then guess which way the branch is going to go, based upon an excellent resource capable of prediction—the programmer. Only in the case of a programmer's wrong prediction would a pipeline stall occur. Furthermore, compilers will take advantage of branch prediction when they detect loops.

It is either obvious, or uncertain, at the time the program is written which way the branches will branch most often during execution. If the likely branch outcome is obvious, the type of branch to use will be obvious. In the cases where runtime factors determine

the branch path, the branch types can be selected to reduce the time through the longest path or to reduce the average path time.

Compare and branch. The compare and branch instructions support integers and ordinals. The CPU compares two operands; the result determines the branch taken. This frequently used operation is one instruction that increases performance and improves code density. The 80960 provides all combinations of branch conditions, in addition to branch-on-bit instructions.

Instruction cache

As processors increase in speed, the traffic between processor and memory becomes a significant performance bottleneck. To effectively reduce this bottleneck, we incorporated an instruction cache within the processor.

An on-chip instruction cache is highly desirable for two reasons. Caching instructions on chip greatly reduces system bus loading and the criticality of the system's memory access speed in a parallel execution environment. However, an instruction cache plays an additional role. The only way to cause multiple instructions to execute simultaneously is to decode multiple instructions simultaneously. An on-chip instruction cache gives instruction decode the capability of looking downstream and decoding and dispatching multiple instructions simultaneously for parallel execution.

The advantage of an instruction cache over a pre-fetch queue, a technique used in most high-performance microprocessors to date, is that a queue does not reduce the memory traffic for instructions. It only attempts to distribute the traffic more efficiently. A cache's most obvious effect occurs with execution loops, common in embedded control applications. After a loop is first executed, successive iterations of the loop generate no memory references for instruction fetches. Likewise, when a small, low-level procedure concludes and executes a RET instruction, the code for the high-level routine to which it is returning likely still resides in the cache. Thus, we reduce the sensitivity of instruction execution speed to slow memory and free valuable bus bandwidth for other operations.

Having an instruction cache requires special considerations in applications that employ self-modifying code or uploadable programs. In general, embedded applications are unaffected. However, for 80960 chips targeted at embedded applications in which volatile code exists, we will provide implementation-specific cache features. For example, implementations could provide a bus input pin that prohibits the data being read from being cached, a method for flushing the cache, a transparent instruction cache, a cache disable bit, or some other feature tuned to the application.

To allow implementations of the 80960 latitude in the amount and type of cache provided, the architecture does not specify the instruction cache parameterization.

User-supervisor protection

The architecture provides a mode and stack switching mechanism called the user-supervisor protection model. This protection model allows a system design in which the kernel code and data reside in the same address space as the user code and data. However, the access to the kernel procedures (called supervisor procedures) occurs through a specified interface. A data structure called the System Procedure Table provides this interface (Figure 5).

The 80960 references the System Procedure Table when a System Call (CALLS) instruction executes. This call is similar to a local call, except that the processor gets the location of the called procedure from the System Procedure Table. Figure 6 illustrates the use of the CALLS instruction. CALLS requires a procedure-number operand that is used as an index into the table.

The System Procedure Table entry referenced by CALLS specifies an entry pointer and an entry type for the called procedure. The 80960 invokes two types of system procedures, *local* and *supervisor*. A procedure that is specified as a local procedure is invoked as if it were called by the CALL or CALLX instructions, except that the processor gets the entry point of the called procedure from the System Procedure Table. Referencing a supervisor procedure, on the other hand, switches the processor to the supervisor execution mode and to the supervisor stack.

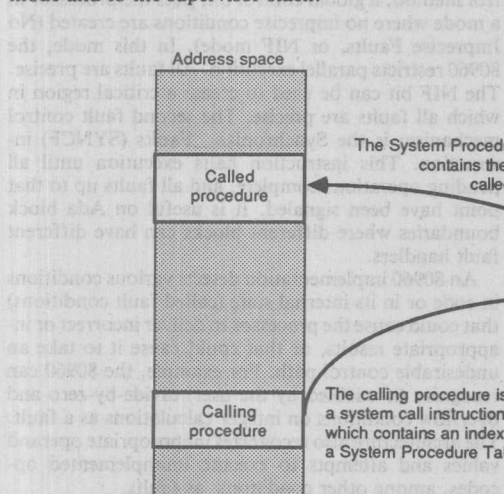


Figure 6. Example of a system procedure call.

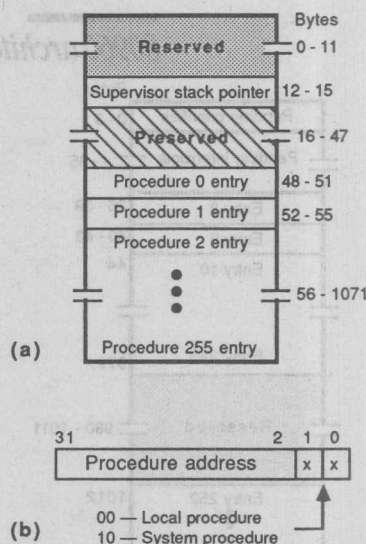


Figure 5. Structure of the System Procedure Table (a) and a procedure entry (b).

Real-time kernel procedures in the supervisor mode execute using a different stack than the one used to execute application programs procedures. Special, supervisor-only instructions also execute in supervisor mode. The MODPC instruction (used to change the processor priority) is always a supervisor instruction. Instruction set extensions that control on-chip hardware are also likely to be restricted to the supervisor mode.

80960 architecture

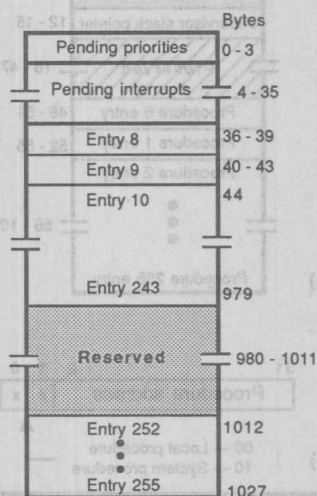


Figure 7. Structure of the interrupt table.

The processor remains in the supervisor mode until the procedure that caused the original mode switch performs a return. Switching stacks and protecting against stack corruption help maintain the integrity of the kernel. For example, the mechanism allows access to system debugging software or a system monitor even if the application crashes.

Interrupts

The 80960 contains a priority interrupt model and a mechanism for queueing pending interrupt requests without user program intervention. When an interrupt is signaled and its priority is higher than the current processor priority, the CPU invokes an interrupt handler and the processor priority changes to that of the interrupt. Otherwise, the 80960 saves the interrupt for service until it becomes the highest priority request pending.

The interrupt table seen in Figure 7 holds the 32-bit pending priorities field, the 256-bit pending interrupt field, and the 248 interrupt vectors. Within each processor priority the 80960 contains eight vectors, eight associated pending interrupt bits, and one pending priority bit. The pending priorities field indicates the priorities at which pending interrupts await. The pending interrupt field indicates exactly which requested interrupts have not yet been serviced.

A pending priority bit is simply the OR of all eight pending interrupt bits at a particular priority. This field optimizes checking for pending interrupts by the processor. When an interrupt request will not be serviced

immediately, the 80960 sets the bit in the pending interrupt field associated with the request. It also sets the pending priorities bit associated with the priority of the request. When the running priority of the processor drops below that of the pending interrupt, the 80960 services the interrupt and clears the associated pending bit. The CPU also clears the associated pending priority bit if appropriate.

Faults

Processors use fault mechanisms to handle exceptions or errant conditions that a program may or may not be capable of correcting. We defined the 80960's fault mechanism for an environment in which parallel or out-of-order execution occurs. When a fault is generated, the processor calls the appropriate fault handler. The 80960 automatically provides the handler with an extensive set of information about the faulting condition for correction or analysis.

It is possible that when a fault is detected not enough information would exist to determine the exact instruction that faulted. For example, when multiple instructions execute in one clock cycle, multiple faults could occur in a single clock cycle. This "imprecise" condition could generate a fault that we call imprecise. A tightly coupled fault handler may be able to recover proper program execution when an imprecise fault occurs. Precise faults are those from which recovery is easy.

The 80960 provides two controls over the generation of imprecise conditions and faults. The first fault control method, a global control bit, puts the processor in a mode where no imprecise conditions are created (No Imprecise Faults, or NIF mode). In this mode, the 80960 restricts parallel execution. All faults are precise. The NIF bit can be used to create a critical region in which all faults are precise. The second fault control mechanism is the Synchronize_Faults (SYNCF) instruction. This instruction halts execution until all pending operations complete, and all faults up to that point have been signaled. It is useful on Ada block boundaries where different blocks can have different fault handlers.

An 80960 implementation detects various conditions in code or in its internal state (called fault conditions) that could cause the processor to deliver incorrect or inappropriate results, or that could cause it to take an undesirable control path. For example, the 80960 can recognize (if enabled by the user) divide-by-zero and overflow conditions on integer calculations as a fault. The architecture also recognizes inappropriate operand values and attempts to execute unimplemented opcodes, among other conditions, as faults.

When a fault is detected, the system processes it immediately and independently of the program or handler that is executing at the time. The fault mechanism is similar to that used by the interrupts. Several fault

types exist, in which the fault type determines which entry in the Fault Table (Figure 8) is invoked for a particular fault. The Fault Table contains one entry for each fault type. The entry defines a particular fault handler routine as a local procedure or a system procedure. When the fault handler is a local procedure, the Fault Table entry contains the address of the procedure entry point. When the fault handler is a system procedure, the Fault Table entry contains the system procedure number, which selects the correct entry point from the System Procedure Table described earlier.

Figure 9 describes the fault record, which is the information provided to a fault handler when a fault occurs. Table 4 on page 76 summarizes the fault types

and subtypes that are currently defined in the 80960 architecture. As extensions to the architecture that consume additional fault types become available, the encoding of fault types and subtypes will occur in such a way that every implementation capable of recognizing similar faulting conditions encodes them identically. For example, the 80960KB adds the floating-point faults (fault type 4). Any other 80960 implementations that also recognize floating-point faults also encode them as fault type 4.

Debug support

Another objective of the architecture is to support software debugging and tracing. A trace-controls register enables most of this support. The trace controls detect any combination of the following events:

- Instruction execution (single step),
- Execution of a Taken Branch instruction,
- Execution of a Call instruction,
- Execution of a Return instruction,

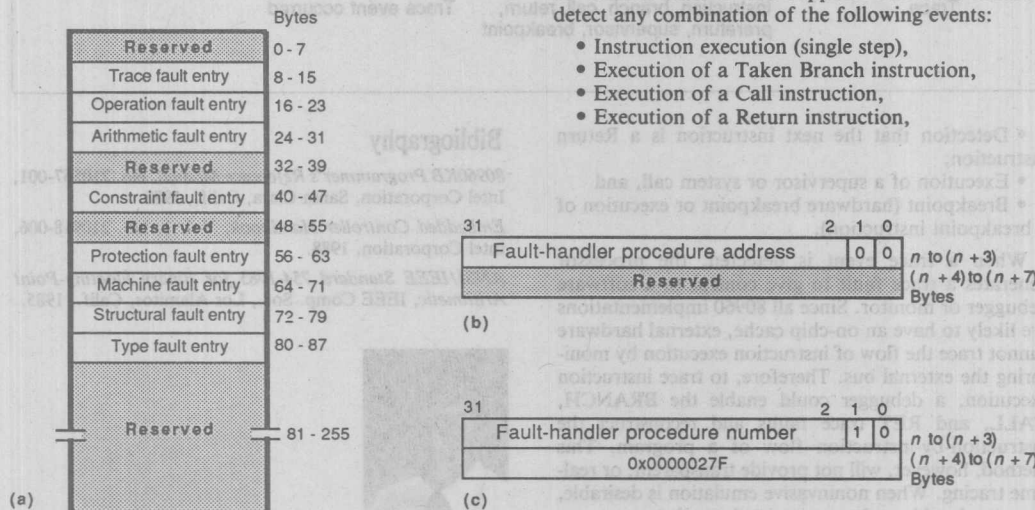


Figure 8. Structure of the fault table (a); an entry to reference a local procedure (b); and an entry to reference a system procedure (c).

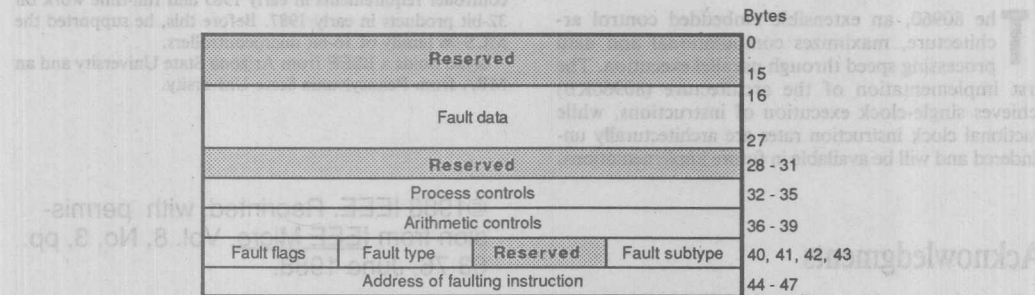


Figure 9. Fault record information. The return pointer r2 is also provided.

80960 architecture

Table 4. Fault types and subtypes.

Fault type	Fault Subtypes	Comments
Arithmetic Constraint Protection Machine Type	Overflow, underflow Range Length Bad access Mismatch	Integer overflows/ divide by zero If FAULT_IF taken Procedure # in CALLS out of range Memory access failed to complete Tried to execute supervisor instruction in nonsupervisor mode
Operation	Invalid opcode, Invalid operand	Tried to execute invalid opcode, or an operand in a valid opcode was invalid
Trace	Instruction, branch, call, return, prereturn, supervisor, breakpoint	Trace event occurred

- Detection that the next instruction is a Return instruction,
- Execution of a supervisor or system call, and
- Breakpoint (hardware breakpoint or execution of a breakpoint instruction).

When a trace event is detected, the processor generates a trace fault to give control to a software debugger or monitor. Since all 80960 implementations are likely to have an on-chip cache, external hardware cannot trace the flow of instruction execution by monitoring the external bus. Therefore, to trace instruction execution, a debugger could enable the BRANCH, CALL, and RET trace faults and reconstruct the instruction-by-instruction flow of a program. This method, however, will not provide transparent, or real-time tracing. When noninvasive emulation is desirable, the user should employ an in-circuit emulator.

The 80960, an extensible embedded control architecture, maximizes computational and data processing speed through parallel execution. The first implementation of the architecture (80960KB) achieves single-clock execution of instructions, while fractional clock instruction rates are architecturally unhindered and will be available in future implementations.

Acknowledgments

Too many people contributed to the 80960 effort to list them here. However, I relied upon the following, either in person or through their writings, in developing this article: Dave Budde, Glen Hinton, Mike Imel, Konrad Lai, Glenford J. Myers, Lew Pacely, Fred Pollack, Rob Riches, Frank Smith, and Randy Steck.

Bibliography

- 80960KB Programmer's Reference Manual, No. 210567-001, Intel Corporation, Santa Clara, Calif., 1988.
- Embedded Controller Handbook, Vol. 1, No. 210918-006, Intel Corporation, 1988.
- ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic, IEEE Comp. Soc., Los Alamitos, Calif., 1985.



David P. Ryan, a senior applications engineer in Intel's embedded controller operation, supervises the Arizona-based 32-bit applications team. He began work on 32-bit embedded controller requirements in early 1985 and full-time work on 32-bit products in early 1987. Before this, he supported the MCS-96 family of 16-bit microcontrollers.

Ryan holds a BSEE from Arizona State University and an MBA from Pennsylvania State University.

©1988 IEEE. Reprinted, with permission from IEEE Micro, Vol. 8, No. 3, pp. 63-76, June 1988.

General Microcontroller Application Notes

9

9

General Microcontroller Application Notes

DESIGNING MICROCONTROLLER SYSTEMS FOR ELECTRICALLY NOISY ENVIRONMENTS

CONTENTS

PAGE

SYMPTOMS OF NOISE PROBLEMS	9-3
TYPES AND SOURCES OF ELECTRICAL NOISE	9-3
Supply Line Transients	9-3
EMP and RFI	9-3
ESD	9-4
Ground Noise	9-4
"RADIATED" AND "CONDUCTED" NOISE	9-4
SIMULATING THE ENVIRONMENT	9-5
TYPES OF FAILURES AND FAILURE MECHANISMS	9-5
THE GAME PLAN	9-6
CURRENT LOOPS	9-6
SHIELDING	9-7
Shielding Against Capacitive Coupling	9-7
Shielding Against Inductive Coupling	9-7
RF Shielding	9-10
GROUNDING	9-11
Safety Ground	9-11
Signal Ground	9-12
Practical Grounding	9-13
Braided Cable	9-13
POWER SUPPLY DISTRIBUTION AND DECOUPLING	9-15
Selecting the Value of the Decoupling Cap	9-16
The Case for On-Board Voltage Regulation	9-17
RECOVERING GRACEFULLY FROM A SOFTWARE UPSET	9-17
SPECIAL PROBLEM AREAS	9-19
ESD	9-19
The Automotive Environment	9-20
PARTING THOUGHTS	9-22
REFERENCES	9-23

Digital circuits are often thought of as being immune to noise problems, but really they're not. Noises in digital systems produce software upsets: program jumps to apparently random locations in memory. Noise-induced glitches in the signal lines can cause such problems, but the supply voltage is more sensitive to glitches than the signal lines.

Severe noise conditions, those involving electrostatic discharges, or as found in automotive environments, can do permanent damage to the hardware. Electrostatic discharges can blow a crater in the silicon. In the automotive environment, in ordinary operation, the "12V" power line can show + and -400V transients.

This Application Note describes some electrical noises and noise environments. Design considerations, along the lines of PCB layout, power supply distribution and decoupling, and shielding and grounding techniques, that may help minimize noise susceptibility are reviewed. Special attention is given to the automotive and ESD environments.

Symptoms of Noise Problems

Noise problems are not usually encountered during the development phase of a microcontroller system. This is because benches rarely simulate the system's intended environment. Noise problems tend not to show up until the system is installed and operating in its intended environment. Then, after a few minutes or hours of normal operation the system finds itself someplace out in left field. Inputs are ignored and outputs are gibberish. The system may respond to a reset, or it may have to be turned off physically and then back on again, at which point it commences operating as though nothing had happened. There may be an obvious cause, such as an electrostatic discharge from somebody's finger to a keyboard or the upset occurs every time a copier machine is turned on or off. Or there may be no obvious cause, and nothing the operator can do will make the upset repeat itself. But a few minutes, or a few hours, or a few days later it happens again.

One symptom of electrical noise problems is randomness, both in the occurrence of the problem and in what the system does in its failure. All operational upsets that occur at seemingly random intervals are not necessarily caused by noise in the system. Marginal VCC, inadequate decoupling, rarely encountered software conditions, or timing coincidences can produce upsets that seem to occur randomly. On the other hand, some noise sources can produce upsets downright periodically. Nevertheless, the more difficult it is to characterize an upset as to cause and effect, the more likely it is to be a noise problem.

Types and Sources of Electrical Noise

The name given to electrical noises other than those that are inherent in the circuit components (such as thermal noise) is EMI: electromagnetic interference. Motors, power switches, fluorescent lights, electrostatic discharges, etc., are sources of EMI. There is a veritable alphabet soup of EMI types, and these are briefly described below.

SUPPLY LINE TRANSIENTS

Anything that switches heavy current loads onto or off of AC or DC power lines will cause large transients in these power lines. Switching an electric typewriter on or off, for example, can put a 1000V spike onto the AC power lines.

The basic mechanism behind supply line transients is shown in Figure 1. The battery represents any power source, AC or DC. The coils represent the line inductance between the power source and the switchable loads R1 and R2. If both loads are drawing current, the line current flowing through the line inductance establishes a magnetic field of some value. Then, when one of the loads is switched off, the field due to that component of the line current collapses, generating transient voltages, $v = L(di/dt)$, which try to maintain the current at its original level. That's called an "inductive kick." Because of contact bounce, transients are generated whether the switch is being opened or closed, but they're worse when the switch is being opened.

An inductive kick of one type or another is involved in most line transients, including those found in the automotive environment. Other mechanisms for line transients exist, involving noise pickup on the lines. The noise voltages are then conducted to a susceptible circuit right along with the power.

EMP AND RFI

Anything that produces arcs or sparks will radiate electromagnetic pulses (EMP) or radio-frequency interference (RFI).

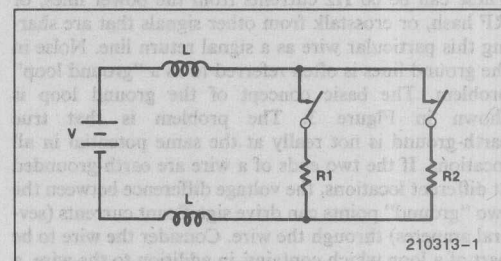


Figure 1. Supply Line Transients

Spark discharges have probably caused more software upsets in digital equipment than any other single noise source. The upsetting mechanism is the EMP produced by the spark. The EMP induces transients in the circuit, which are what actually cause the upset.

Arcs and sparks occur in automotive ignition systems, electric motors, switches, static discharges, etc. Electric motors that have commutator bars produce an arc as the brushes pass from one bar to the next. DC motors and the "universal" (AC/DC) motors that are used to power hand tools are the kinds that have commutator bars. In switches, the same inductive kick that puts transients on the supply lines will cause an opening or closing switch to throw a spark.

ESD

Electrostatic discharge (ESD) is the spark that occurs when a person picks up a static charge from walking across a carpet, and then discharges it into a keyboard, or whatever else can be touched. Walking across a carpet in a dry climate, a person can accumulate a static voltage of 35kV. The current pulse from an electrostatic discharge has an extremely fast risetime — typically, 4A/ns. Figure 2 shows ESD waveforms that have been observed by some investigators of ESD phenomena.

It is enlightening to calculate the $L(di/dt)$ voltage required to drive an ESD current pulse through a couple of inches of straight wire. Two inches of straight wire has about 50 nH of inductance. That's not very much, but using 50 nH for L and 4A/ns for di/dt gives an $L(di/dt)$ drop of about 200V. Recent observations by W.M. King suggest even faster risetimes (Figure 2b) and the occurrence of multiple discharges during a single discharge event.

Obviously, ESD-sensitivity needs to be considered in the design of equipment that is going to be subjected to it, such as office equipment.

GROUND NOISE

Currents in ground lines are another source of noise. These can be 60 Hz currents from the power lines, or RF hash, or crosstalk from other signals that are sharing this particular wire as a signal return line. Noise in the ground lines is often referred to as a "ground loop" problem. The basic concept of the ground loop is shown in Figure 3. The problem is that true earth-ground is not really at the same potential in all locations. If the two ends of a wire are earth-grounded at different locations, the voltage difference between the two "ground" points can drive significant currents (several amperes) through the wire. Consider the wire to be part of a loop which contains, in addition to the wire, a voltage source that represents the difference in potential between the two ground points, and you have

the classical "ground loop." By extension, the term is used to refer to any unwanted (and often unexpected) currents in a ground line.

"Radiated" and "Conducted" Noise

Radiated noise is noise that arrives at the victim circuit in the form of electromagnetic radiation, such as EMP and RFI. It causes trouble by inducing extraneous voltages in the circuit. Conducted noise is noise that arrives at the victim circuit already in the form of an extraneous voltage, typically via the AC or DC power lines.

One defends against radiated noise by care in designing layouts and the use of effective shielding techniques. One defends against conducted noise with filters and

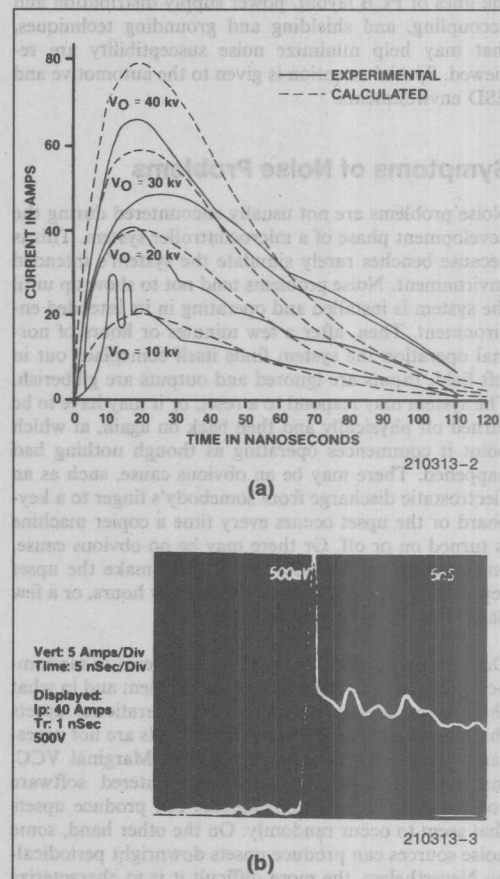


Figure 2. Waveforms of Electrostatic Discharge Currents From a Hand-Held Metallic Object

suppressors, although layouts and grounding techniques are important here, too.

Simulating the Environment

Addressing noise problems after the design of a system has been completed is an expensive proposition. The ill will generated by failures in the field is not cheap either. It's cheaper in the long run to invest a little time and money in learning about noise and noise simulation equipment, so that controlled tests can be made on the bench as the design is developing.

Simulating the intended noise environment is a two-step process: First you have to recognize what the noise environment is, that is, you have to know what kinds of electrical noises are present, and which of them are going to cause trouble. Don't ignore this first step, because it's important. If you invest in an induction coil spark generator just because your application is automotive, you'll be straining at the gnat and swallowing the camel. Spark plug noise is the least of your worries in that environment.

The second step is to generate the electrical noise in a controlled manner. This is usually more difficult than first imagined; one first imagines the simulation in terms of a waveform generator and a few spare parts, and then finds that a wideband power amplifier with a 200V dynamic range is also required. A good source of information on who supplies what noise-simulating equipment is the 1981 "ITEM" Directory and Design Guide (Reference 6).

Types of Failures and Failure Mechanisms

A major problem that EMI can cause in digital systems is intermittent operational malfunction. These software upsets occur when the system is in operation at the time an EMI source is activated, and are usually characterized by a loss of information or a jump in the execution

of the program to some random location in memory. The person who has to iron out such problems is tempted to say the program counter went crazy. There is usually no damage to the hardware, and normal operation can resume as soon as the EMI has passed or the source is de-activated. Resuming normal operation usually requires manual or automatic reset, and possibly re-entering of lost information.

Electrostatic discharges from operating personnel can cause not only software upsets, but also permanent ("hard") damage to the system. For this to happen the system doesn't even have to be in operation. Sometimes the permanent damage is latent, meaning the initial damage may be marginal and require further aggravation through operating stress and time before permanent failure takes place. Sometimes too the damage is hidden.

One ESD-related failure mechanism that has been identified has to do with the bias voltage on the substrate of the chip. On some CPU chips the substrate is held at $-2.5V$ by a phase-shift oscillator working into a capacitor/diode clamping circuit. This is called a "charge pump" in chip-design circles. If the substrate wanders too far in either direction, program read errors are noted. Some designs have been known to allow electrostatic discharge currents to flow directly into port pins of an 8048. The resulting damage to the oxide causes an increase in leakage current, which loads down the charge pump, reducing the substrate voltage to a marginal or unacceptable level. The system is then unreliable or completely inoperative until the CPU chip is replaced. But if the CPU chip was subjected to a discharge spark once, it will eventually happen again.

Chips that have a grounded substrate, such as the 8748, can sometimes sustain some oxide damage without actually becoming inoperative. In this case the damage is present, and the increased leakage current is noted; however, since the substrate voltage retains its design value, the damage is largely hidden.

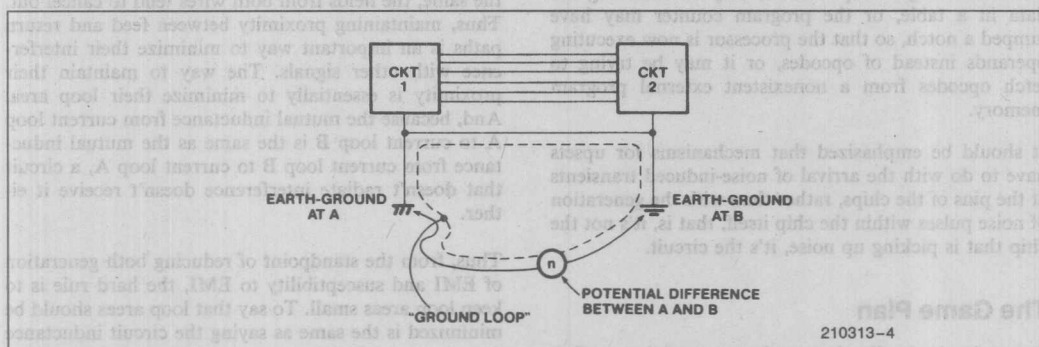


Figure 3. What a Ground Loop Is

It must therefore be recognized that connecting port pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges, makes an extremely dangerous configuration. It doesn't make any difference what CPU chip is being used, or who makes it. If it connects unprotected to a keyboard, it will eventually be destroyed. Designing for an ESD-environment will be discussed further on.

We might note here that MOS chips are not the only components that are susceptible to permanent ESD damage. Bipolar and linear chips can also be damaged in this way. PN junctions are subject to a hard failure mechanism called thermal secondary breakdown, in which a current spike, such as from an electrostatic discharge, causes microscopically localized spots in the junction to approach melt temperatures. Low power TTL chips are subject to this type of damage, as are op-amps. Op-amps, in addition, often carry on-chip MOS capacitors which are directly across an external pin combination, and these are susceptible to dielectric breakdown.

We return now to the subject of software upsets. Noise transients can upset the chip through any pin, even an output pin, because every pin on the chip connects to the substrate through a pn junction. However, the most vulnerable pin is probably the VCC line, since it has direct access to all parts of the chip: every register, gate, flip-flop and buffer.

The menu of possible upset mechanisms is quite lengthy. A transient on the substrate at the wrong time will generally cause a program read error. A false level at a control input can cause an extraneous or misdirected opcode fetch. A disturbance on the supply line can flip a bit in the program counter or instruction register. A short interruption or reversal of polarity on the supply line can actually turn the processor off, but not long enough for the power-up reset capacitor to discharge. Thus when the transient ends, the chip starts up again without a reset.

A common failure mode is for the processor to lock itself into a tight loop. Here it may be executing the data in a table, or the program counter may have jumped a notch, so that the processor is now executing operands instead of opcodes, or it may be trying to fetch opcodes from a nonexistent external program memory.

It should be emphasized that mechanisms for upsets have to do with the arrival of noise-induced transients at the pins of the chips, rather than with the generation of noise pulses within the chip itself, that is, it's not the chip that is picking up noise, it's the circuit.

The Game Plan

Prevention is usually cheaper than suppression, so first we'll consider some preventive methods that might help

to minimize the generation of noise voltages in the circuit. These methods involve grounding, shielding, and wiring techniques that are directed toward the mechanisms by which noise voltages are generated in the circuit. We'll also discuss methods of decoupling. Then we'll look at some schemes for making a graceful recovery from upsets that occur in spite of preventive measures. Lastly, we'll take another look at two special problem areas: electrostatic discharges and the automotive environment.

Current Loops

The first thing most people learn about electricity is that current won't flow unless it can flow in a closed loop. This simple fact is sometimes temporarily forgotten by the overworked engineer who has spent the past several years mastering the intricacies of the DO loop, the timing loop, the feedback loop, and maybe even the ground loop. The simple current loop probably owes its apparent demise to the invention of the ground symbol. By a stroke of the pen one avoids having to draw the return paths of most of the current loops in the circuit. Then "ground" turns into an infinite current sink, so that any current that flows into it is gone and forgotten. Forgotten it may be, but it's not gone. It must return to its source, so that its path will by all the laws of nature form a closed loop.

The physical geometry of a given current loop is the key to why it generates EMI, why it's susceptible to EMI, and how to shield it. Specifically, it's the area of the loop that matters.

Any flow of current generates a magnetic field whose intensity varies inversely to the distance from the wire that carries the current. Two parallel wires conducting currents $+I$ and $-I$ (as in signal feed and return lines) would generate a nonzero magnetic field near the wires, where the distance from a given point to one wire is noticeably different from the distance to the other wire, but farther away (relative to the wire spacing), where the distances from a given point to either wire are about the same, the fields from both wires tend to cancel out. Thus, maintaining proximity between feed and return paths is an important way to minimize their interference with other signals. The way to maintain their proximity is essentially to minimize their loop area. And, because the mutual inductance from current loop A to current loop B is the same as the mutual inductance from current loop B to current loop A, a circuit that doesn't radiate interference doesn't receive it either.

Thus, from the standpoint of reducing both generation of EMI and susceptibility to EMI, the hard rule is to keep loop areas small. To say that loop areas should be minimized is the same as saying the circuit inductance

should be minimized. Inductance is by definition the constant of proportionality between current and the magnetic field it produces: $\phi = LI$. Holding the feed and return wires close together so as to promote field cancellation can be described either as minimizing the loop area or as minimizing L . It's the same thing.

Shielding

There are three basic kinds of shields: shielding against capacitive coupling, shielding against inductive coupling, and RF shielding. Capacitive coupling is electric field coupling, so shielding against it amounts to shielding against electric fields. As will be seen, this is relatively easy. Inductive coupling is magnetic field coupling, so shielding against it is shielding against magnetic fields. This is a little more difficult. Strangely enough, this type of shielding does not in general involve the use of magnetic materials. RF shielding, the classical "metallic barrier" against all sorts of electromagnetic fields, is what most people picture when they think about shielding. Its effectiveness depends partly on the selection of the shielding material, but mostly, as it turns out, on the treatment of its seams and the geometry of its openings.

SHIELDING AGAINST CAPACITIVE COUPLING

Capacitive coupling involves the passage of interfering signals through mutual or stray capacitances that aren't shown on the circuit diagram, but which the experienced engineer knows are there. Capacitive coupling to one's body is what would cause an unstable oscillator to change its frequency when the person reaches his hand over the circuit, for example. More importantly, in a digital system it causes crosstalk in multi-wire cables.

The way to block capacitive coupling is to enclose the circuit or conductor you want to protect in a metal shield. That's called an electrostatic or Faraday shield. If coverage is 100%, the shield does not have to be grounded, but it usually is, to ensure that circuit-to-shield capacitances go to signal reference ground rather than act as feedback and crosstalk elements. Besides, from a mechanical point of view, grounding it is almost inevitable.

A grounded Faraday shield can be used to break capacitive coupling between a noisy circuit and a victim circuit, as shown in Figure 4. Figure 4a shows two circuits capacitively coupled through the stray capacitance between them. In Figure 4b the stray capacitance is intercepted by a grounded Faraday shield, so that interference currents are shunted to ground. For example, a grounded plane can be inserted between PCBs (printed circuit boards) to eliminate most of the capacitive coupling between them.

Another application of the Faraday shield is in the electrostatically shielded transformer. Here, a conducting foil is laid between the primary and secondary coils so as to intercept the capacitive coupling between them. If a system is being upset by AC line transients, this type of transformer may provide the fix. To be effective in this application, the shield must be connected to the greenwire ground.

SHIELDING AGAINST INDUCTIVE COUPLING

With inductive coupling, the physical mechanism involved is a magnetic flux density B from some external interference source that links with a current loop in the victim circuit, and generates a voltage in the loop in accordance with Lenz's law: $v = -NA(dB/dt)$, where in this case $N = 1$ and A is the area of the current loop in the victim circuit.

There are two aspects to defending a circuit against inductive pickup. One aspect is to try to minimize the offensive fields at their source. This is done by minimizing the area of the current loop at the source so as to promote field cancellation, as described in the section on current loops. The other aspect is to minimize the inductive pickup in the victim circuit by minimizing the area of that current loop, since, from Lenz's law, the induced voltage is proportional to this area. So the two aspects really involve the same corrective action: minimize the areas of the current loops. In other words, minimizing the offensiveness of a circuit inherently minimizes its susceptibility.

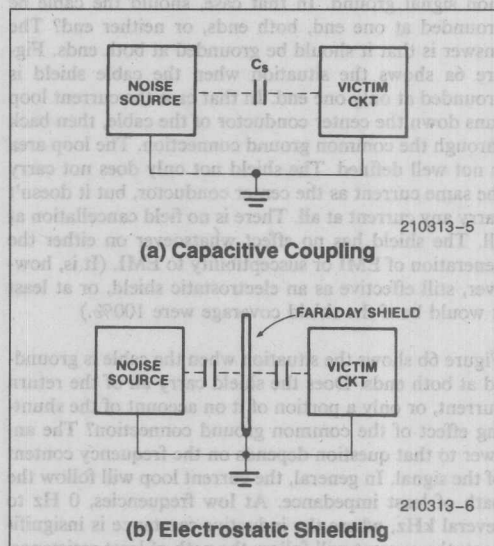


Figure 4. Use of Faraday Shield

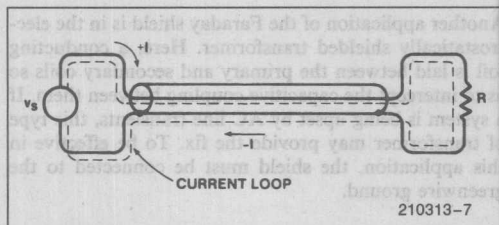


Figure 5. External to the Shield, $\phi = 0$

Shielding against inductive coupling means nothing more nor less than controlling the dimensions of the current loops in the circuit. We must look at four examples of this type of "shielding": the coaxial cable, the twisted pair, the ground plane, and the gridded-ground PCB layout.

The Coaxial Cable—Figure 5 shows a coaxial cable carrying a current I from a signal source to a receiving load. The shield carries the same current as the center conductor. Outside the shield, the magnetic field produced by $+I$ flowing in the center conductor is cancelled by the field produced by $-I$ flowing in the shield. To the extent that the cable is ideal in producing zero external magnetic field, it is immune to inductive pickup from external sources. The cable adds effectively zero area to the loop. This is true only if the shield carries the same current as the center conductor.

In the real world, both the signal source and the receiving load are likely to have one end connected to a common signal ground. In that case, should the cable be grounded at one end, both ends, or neither end? The answer is that it should be grounded at both ends. Figure 6a shows the situation when the cable shield is grounded at only one end. In that case the current loop runs down the center conductor of the cable, then back through the common ground connection. The loop area is not well defined. The shield not only does not carry the same current as the center conductor, but it doesn't carry any current at all. There is no field cancellation at all. The shield has no effect whatsoever on either the generation of EMI or susceptibility to EMI. (It is, however, still effective as an electrostatic shield, or at least it would be if the shield coverage were 100%.)

Figure 6b shows the situation when the cable is grounded at both ends. Does the shield carry all of the return current, or only a portion of it on account of the shunting effect of the common ground connection? The answer to that question depends on the frequency content of the signal. In general, the current loop will follow the path of least impedance. At low frequencies, 0 Hz to several kHz, where the inductive reactance is insignificant, the current will follow the path of least resistance. Above a few kHz, where inductive reactance predominates, the current will follow the path of least inductance. The path of least inductance is the path of

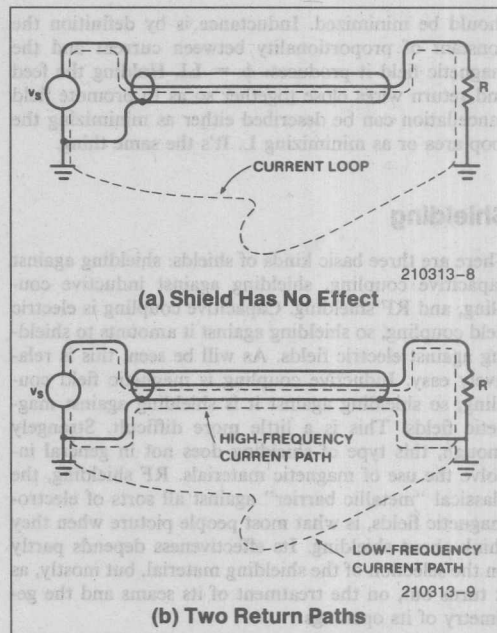


Figure 6. Use of Coaxial Cable

minimum loop area. Hence, for higher frequencies the shield carries virtually the same current as the center conductor, and is therefore effective against both generation and reception of EMI.

Note that we have now introduced the famous "ground loop" problem, as shown in Figure 7a. Fortunately, a digital system has some built-in immunity to moderate ground loop noise. In a noisy environment, however, one can break the ground loop, and still maintain the shielding effectiveness of the coaxial cable, by inserting an optical coupler, as shown in Figure 7b. What the optical coupler does, basically, is allow us to re-define the signal source as being ungrounded, so that that end of the cable need not be grounded, and still lets the shield carry the same current as the center conductor. Obviously, if the signal source weren't grounded in the first place, the optical coupler wouldn't be needed.

The Twisted Pair—A cheaper way to minimize loop area is to run the feed and return wires right next to each other. This isn't as effective as a coaxial cable in minimizing loop area. An ideal coaxial cable adds zero area to the loop, whereas merely keeping the feed and return wires next to each other is bound to add a finite area.

However, two things work to make this cheaper method almost as good as a coaxial cable. First, real coaxial cables are not ideal. If the shield current isn't evenly distributed around the center conductor at every cross-

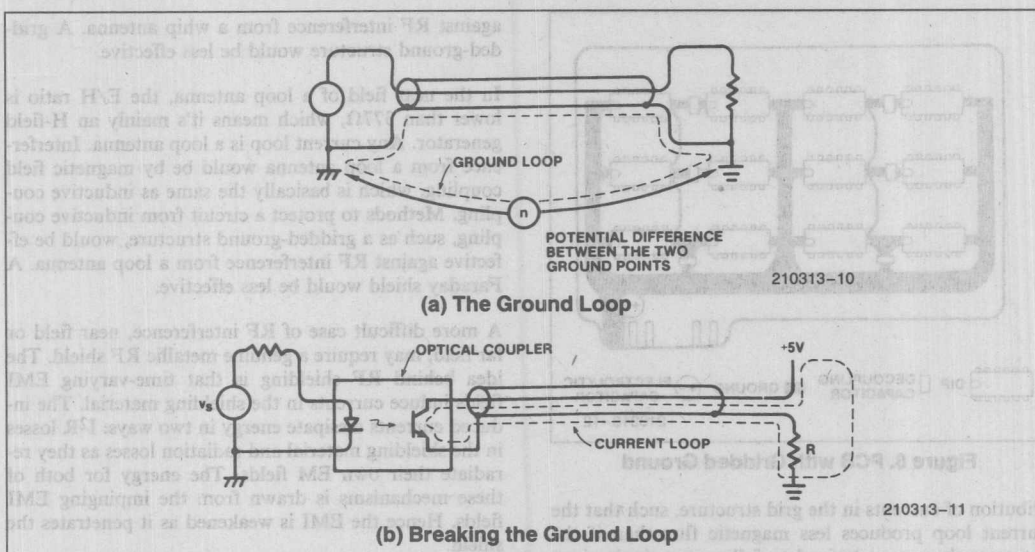


Figure 7. Use of Optical Coupler

section of the cable (it isn't), then field cancellation external to the shield is incomplete. If field cancellation is incomplete, then the effective area added to the loop by the cable isn't zero. Second, in the cheaper method the feed and return wires can be twisted together. This not only maintains their proximity, but the noise picked up in one twist tends to cancel out the noise picked up in the next twist down the line. Thus the "twisted pair" turns out to be about as good a shield against inductive coupling as coaxial cable is.

The twisted pair does not, however, provide electrostatic shielding (i.e., shielding against capacitive coupling). Another operational difference between them is that the coaxial cable works better at higher frequencies. This is primarily because the twisted pair adds more capacitive loading to the signal source than the coaxial cable does. The twisted pair is normally considered useful up to only about 1 MHz, as opposed to near a GHz for the coaxial cable.

The Ground Plane—The best way to minimize loop areas when many current loops are involved is to use a ground plane. A ground plane is a conducting surface that is to serve as a return conductor for all the current loops in the circuit. Normally, it would be one or more layers of a multilayer PCB. All ground points in the circuit go not to a grounded trace on the PCB, but directly to the ground plane. This leaves each current loop in the circuit free to complete itself in whatever configuration yields minimum loop area (for frequencies wherein the ground path impedance is primarily inductive).

Thus, if the feed path for a given signal zigzags its way across the PCB, the return path for this signal is free to zigzag right along beneath it on the ground plane, in such a configuration as to minimize the energy stored in the magnetic field produced by this current loop. Minimal magnetic flux means minimal effective loop area and minimal susceptibility to inductive coupling.

The Gridded-Ground PCB Layout—The next best thing to a ground plane is to lay out the ground traces on a PCB in the form of a grid structure, as shown in Figure 8. Laying horizontal traces on one side of the board and vertical traces on the other side allows the passage of signal and power traces. Wherever vertical and horizontal ground traces cross, they must be connected by a feed-through.

Have we not created here a network of "ground loops"? Yes, in the literal sense of the word, but loops in the ground layout on a PCB are not to be feared. Such inoffensive little loops have never caused as much noise pickup as their avoidance has. Trying to avoid innocent little loops in the ground layout, PCB designers have forced current loops into geometries that could swallow a whale. That is exactly the wrong thing to do.

The gridded ground structure works almost as well as the ground plane, as far as minimizing loop area is concerned. For a given current loop, the primary return path may have to zig once in a while where its feed path zags, but you still get a mathematically optimal dis-

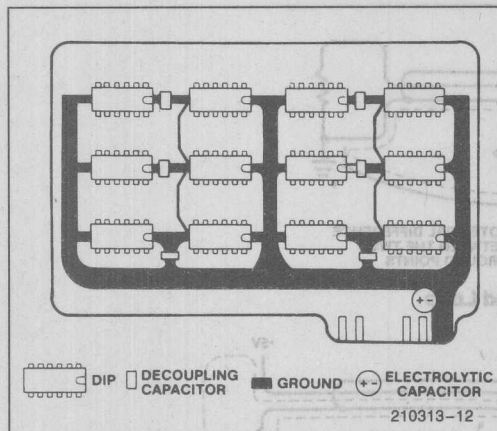


Figure 8. PCB with Gridded Ground

tribution of currents in the grid structure, such that the current loop produces less magnetic flux than if the return path were restrained to follow any single given ground trace. The key to attaining minimum loop areas for all the current loops together is to let the ground currents distribute themselves around the entire area of the board as freely as possible. They want to minimize their own magnetic field. Just let them.

RF SHIELDING

A time-varying electric field generates a time-varying magnetic field, and vice versa. Far from the source of a time-varying EM field, the ratio of the amplitudes of the electric and magnetic fields is always 377Ω . Up close to the source of the fields, however, this ratio can be quite different, and dependent on the nature of the source. Where the ratio is near 377Ω is called the far field, and where the ratio is significantly different from 377Ω is called the near field. The ratio itself is called the wave impedance, E/H.

The near field goes out about 1/6 of a wavelength from the source. At 1 MHz this is about 150 feet, and at 10 MHz it's about 15 feet. That means if an EMI source is in the same room with the victim circuit, it's likely to be a near field problem. The reason this matters is that in the near field an RF interference problem could be almost entirely due to E-field coupling or H-field coupling, and that could influence the choice of an RF shield or whether an RF shield will help at all.

In the near field of a whip antenna, the E/H ratio is higher than 377Ω , which means it's mainly an E-field generator. A wire-wrap post can be a whip antenna. Interference from a whip antenna would be by electric field coupling, which is basically capacitive coupling. Methods to protect a circuit from capacitive coupling, such as a Faraday shield, would be effective

against RF interference from a whip antenna. A gridded-ground structure would be less effective.

In the near field of a loop antenna, the E/H ratio is lower than 377Ω , which means it's mainly an H-field generator. Any current loop is a loop antenna. Interference from a loop antenna would be by magnetic field coupling, which is basically the same as inductive coupling. Methods to protect a circuit from inductive coupling, such as a gridded-ground structure, would be effective against RF interference from a loop antenna. A Faraday shield would be less effective.

A more difficult case of RF interference, near field or far field, may require a genuine metallic RF shield. The idea behind RF shielding is that time-varying EMI fields induce currents in the shielding material. The induced currents dissipate energy in two ways: I^2R losses in the shielding material and radiation losses as they re-radiate their own EM fields. The energy for both of these mechanisms is drawn from the impinging EMI fields. Hence the EMI is weakened as it penetrates the shield.

More formally, the I^2R losses are referred to as absorption loss, and the re-radiation is called reflection loss. As it turns out, absorption loss is the primary shielding mechanism for H-fields, and reflection loss is the primary shielding mechanism for E-fields. Reflection loss, being a surface phenomenon, is pretty much independent of the thickness of the shielding material. Both loss mechanisms, however, are dependent on the frequency (ω) of the impinging EMI field, and on the permeability (μ) and conductivity (σ) of the shielding material. These loss mechanisms vary approximately as follows:

$$\text{reflection loss to an E-field (in dB)} \sim \log \frac{\sigma}{\omega \mu}$$

$$\text{absorption loss to an H-field (in dB)} \sim \sqrt{\omega \sigma \mu}$$

where t is the thickness of the shielding material.

The first expression indicates that E-field shielding is more effective if the shield material is highly conductive, and less effective if the shield is ferromagnetic, and that low-frequency fields are easier to block than high-frequency fields. This is shown in Figure 9.

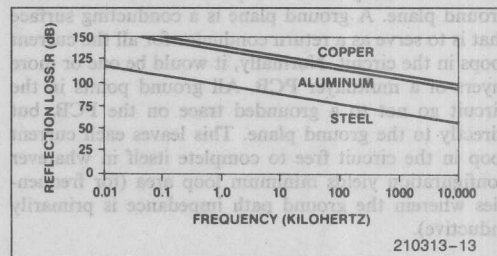


Figure 9. E-Field Shielding

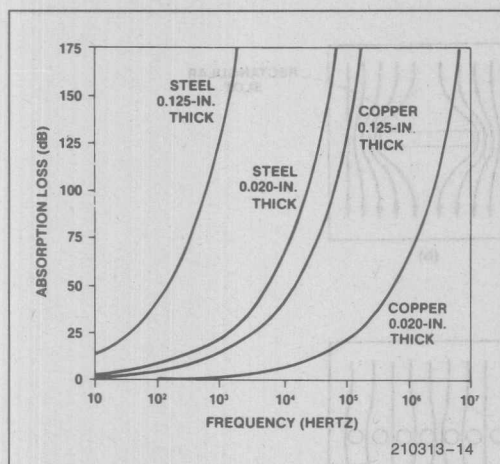


Figure 10. H-Field Shielding

Copper and aluminum both have the same permeability, but copper is slightly more conductive, and so provides slightly greater reflection loss to an E-field. Steel is less effective for two reasons. First, it has a somewhat elevated permeability due to its iron content, and second, as tends to be the case with magnetic materials, it is less conductive.

On the other hand, according to the expression for absorption loss to an H-field, H-field shielding is more effective at higher frequencies and with shield material that has both high conductivity and high permeability. In practice, however, selecting steel for its high permeability involves some compromise in conductivity. But the increase in permeability more than makes up for the decrease in conductivity, as can be seen in Figure 10. This figure also shows the effect of shield thickness.

A composite of E-field and H-field shielding is shown in Figure 11. However, this type of data is meaningful only in the far field. In the near field the EMI could be 90% H-field, in which case the reflection loss is irrelevant. It would be advisable then to beef up the absorption loss, at the expense of reflection loss, by choosing steel. A better conductor than steel might be less expensive, but quite ineffective.

A different shielding mechanism that can be taken advantage of for low frequency magnetic fields is the ability of a high permeability material such as mumetal to divert the field by presenting a very low reluctance path to the magnetic flux. Above a few kHz, however, the permeability of such materials is the same as steel.

In actual fact the selection of a shielding material turns out to be less important than the presence of seams, joints and holes in the physical structure of the enclosure. The shielding mechanisms are related to the induction of currents in the shield material, but the cur-

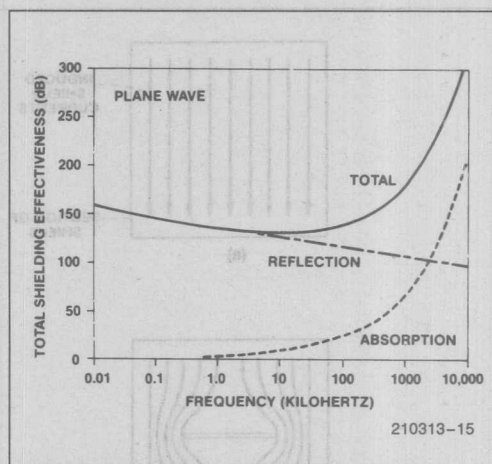


Figure 11. E- and H-Field Shielding

rents must be allowed to flow freely. If they have to detour around slots and holes, as shown in Figure 12, the shield loses much of its effectiveness.

As can be seen in Figure 12, the severity of the detour has less to do with the area of the hole than it does with the geometry of the hole. Comparing Figure 12c with 12d shows that a long narrow discontinuity such as a seam can cause more RF leakage than a line of holes with larger total area. A person who is responsible for designing or selecting rack or chassis enclosures for an EMI environment needs to be familiar with the techniques that are available for maintaining electrical continuity across seams. Information on these techniques is available in the references.

Grounds

There are two kinds of grounds: earth-ground and signal ground. The earth is not an equipotential surface, so earth ground potential varies. That and its other electrical properties are not conducive to its use as a return conductor in a circuit. However, circuits are often connected to earth ground for protection against shock hazards. The other kind of ground, signal ground, is an arbitrarily selected reference node in a circuit—the node with respect to which other node voltages in the circuit are measured.

SAFETY GROUND

The standard 3-wire single-phase AC power distribution system is represented in Figure 13. The white wire is earth-grounded at the service entrance. If a load circuit has a metal enclosure or chassis, and if the black wire develops a short to the enclosure, there will be a shock hazard to operating personnel, unless the enclosure itself is earth-grounded. If the enclosure is earth-

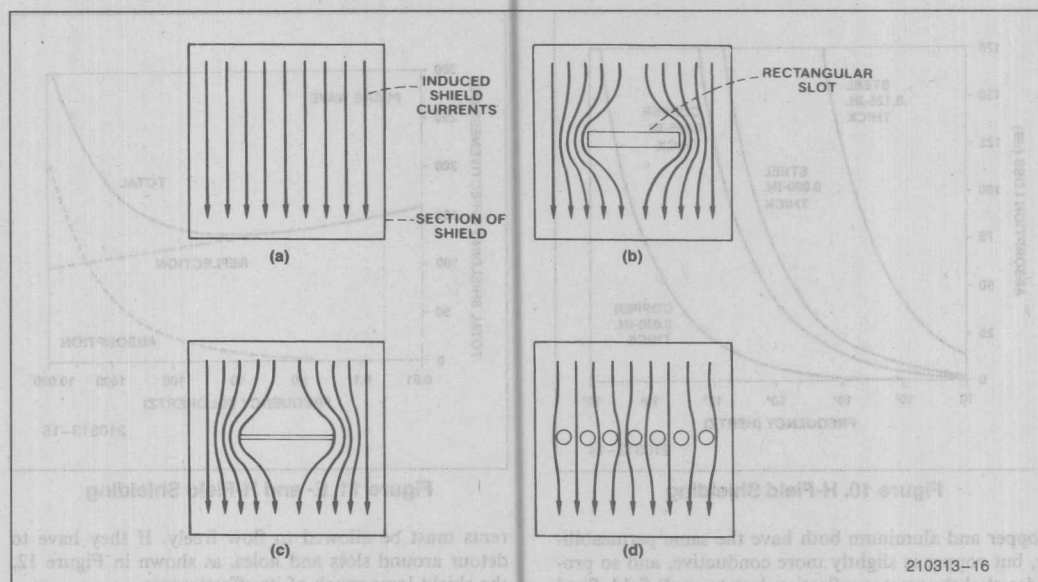


Figure 12. Effect of Shield Discontinuity on Magnetically Induced Shield Current

grounded, a short results in a blown fuse rather than a "hot" enclosure. The earth-ground connection to the enclosure is called a safety ground. The advantage of the 3-wire power system is that it distributes a safety ground along with the power.

Note that the safety-ground wire carries no current, except in case of a fault, so that at least for low frequencies it's at earth-ground potential along its entire length. The white wire, on the other hand, may be several volts off ground, due to the IR drop along its length.

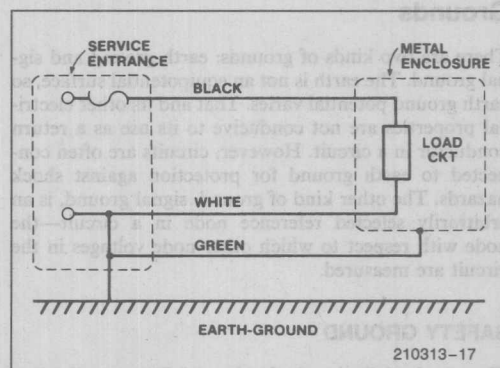


Figure 13. Single-Phase Power Distribution

SIGNAL GROUND

Signal ground is a single point in a circuit that is designated to be the reference node for the circuit. Commonly, wires that connect to this single point are also referred to as "signal ground." In some circles "power supply common" or PSC is the preferred terminology for these conductors. In any case, the manner in which these wires connect to the actual reference point is the basis of distinction among three kinds of signal-ground wiring methods: series, parallel, and multipoint. These methods are shown in Figure 14.

The series connection is pretty common because it's simple and economical. It's the noisiest of the three, however, due to common ground impedance coupling between the circuits. When several circuits share a ground wire, currents from one circuit, flowing through the finite impedance of the common ground line, cause variations in the ground potential of the other circuits. Given that the currents in a digital system tend to be spiked, and that the common impedance is mainly inductive reactance, the variations could be bad enough to cause bit errors in high current or particularly noisy situations.

The parallel connection eliminates common ground impedance problems, but uses a lot of wire. Other disadvantages are that the impedance of the individual ground lines can be very high, and the ground lines themselves can become sources of EMI.

In the multipoint system, ground impedance is minimized by using a ground plane with the various circuits connected to it by very short ground leads. This type of connection would be used mainly in RF circuits above 10 MHz.

PRACTICAL GROUNDING

A combination of series and parallel ground-wiring methods can be used to trade off economic and the various electrical considerations. The idea is to run series connections for circuits that have similar noise properties, and connect them at a single reference point, as in the parallel method, as shown in Figure 15.

In Figure 15, "noisy signal ground" connects to things like motors and relays. Hardware ground is the safety ground connection to chassis, racks, and cabinets. It's a mistake to use the hardware ground as a return path for signal currents because it's fairly noisy (for example, it's the hardware ground that receives an ESD spark) and tends to have high resistance due to joints and seams.

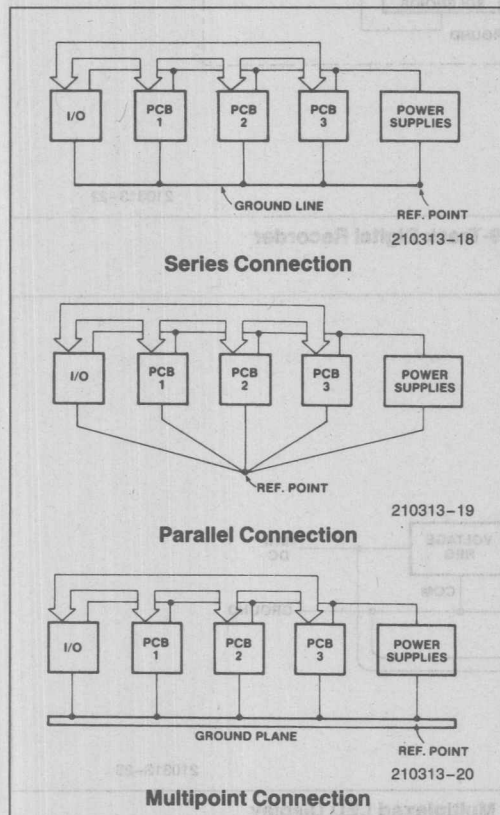


Figure 14. Three Ways to Wire the Grounds

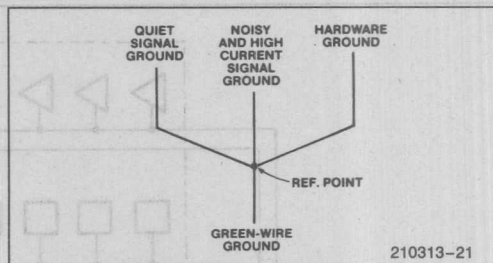


Figure 15. Parallel Connection of Series Grounds

Screws and bolts don't always make good electrical connections because of galvanic action, corrosion, and dirt. These kinds of connections may work well at first, and then cause mysterious maladies as the system ages.

Figure 16 illustrates a grounding system for a 9-track digital tape recorder, showing an application of the series/parallel ground-wiring method.

Figure 17 shows a similar separation of grounds at the PCB level. Currents in multiplexed LED displays tend to put a lot of noise on the ground and supply lines because of the constant switching and changing involved in the scanning process. The segment driver ground is relatively quiet, since it doesn't conduct the LED currents. The digit driver ground is noisier, and should be provided with a separate path to the PCB ground terminal, even if the PCB ground layout is gridded. The LED feed and return current paths should be laid out on opposite sides of the board like parallel flat conductors.

Figure 18 shows right and wrong ways to make ground connections in racks. Note that the safety ground connections from panel to rack are made through ground straps, not panel screws. Rack 1 correctly connects signal ground to rack ground only at the single reference point. Rack 2 incorrectly connects signal ground to rack ground at two points, creating a ground loop around points 1, 2, 3, 4, 1.

Breaking the "electronics ground" connection to point 1 eliminates the ground loop, but leaves signal ground in rack 2 sharing a ground impedance with the relatively noisy hardware ground to the reference point; in fact, it may end up using hardware ground as a return path for signal and power supply currents. This will probably cause more problems than the ground loop.

BRAIDED CABLE

Ground impedance problems can be virtually eliminated by using braided cable. The reduction in impedance is due to skin effect: At higher frequencies the current tends to flow along the surface of a conductor rather

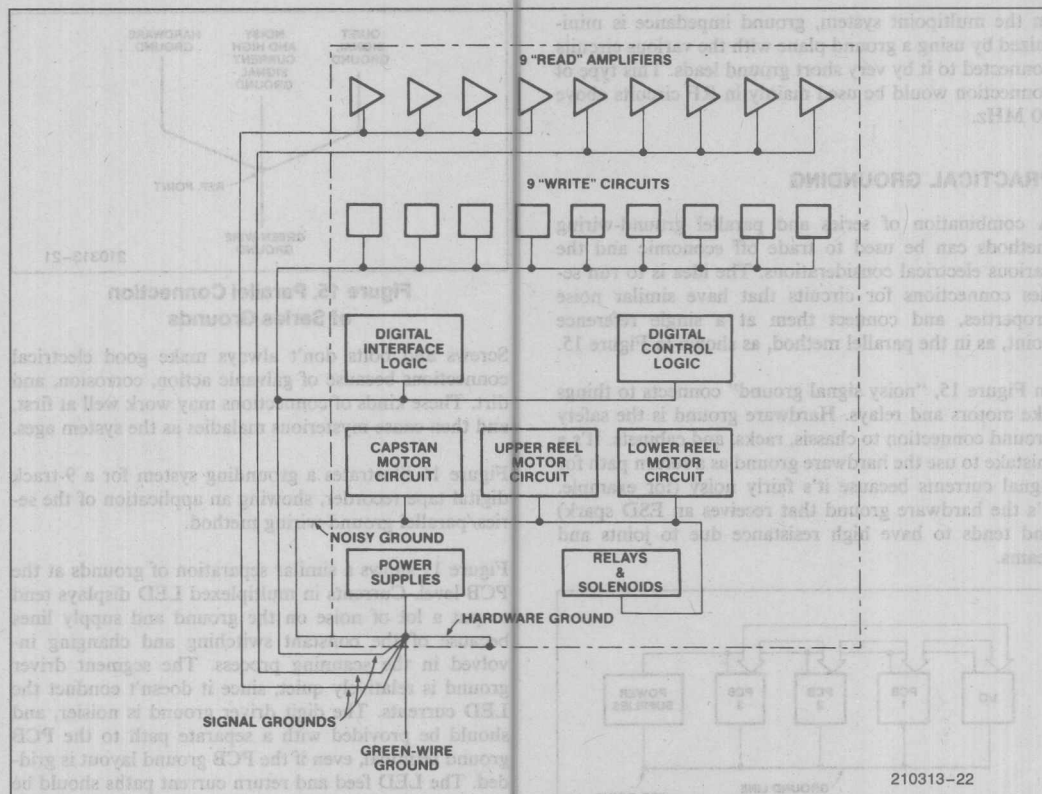


Figure 16. Ground System in a 9-Track Digital Recorder

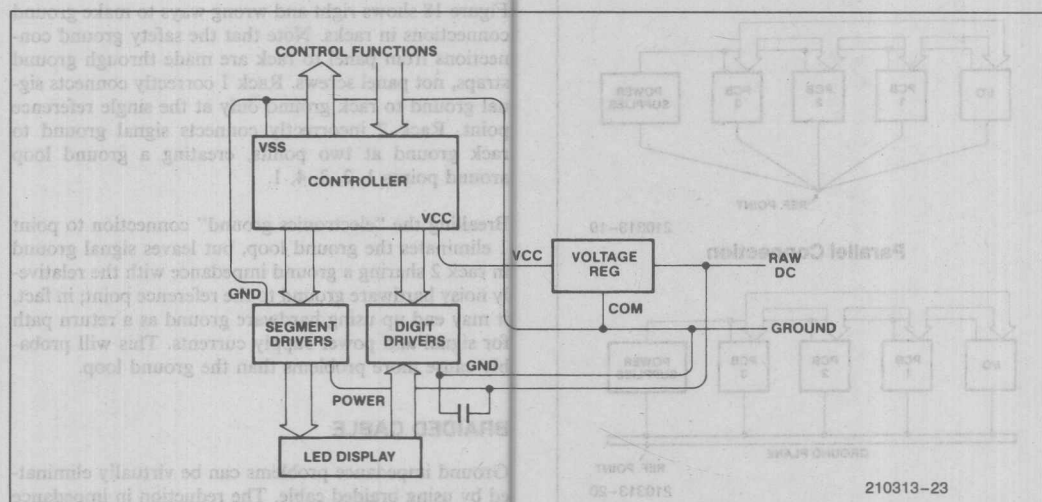


Figure 17. Separate Ground for Multiplexed LED Display

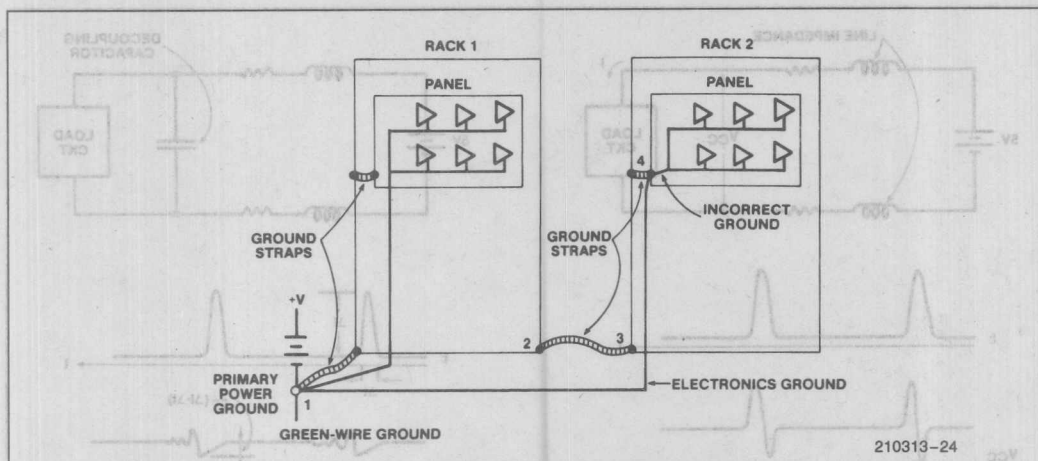


Figure 18. Electronic Circuits Mounted in Equipment Racks Should Have Separate Ground Connections. Rack 1 Shows Correct Grounding, Rack 2 Shows Incorrect Grounding.

than uniformly through its bulk. While this effect tends to increase the impedance of a given conductor, it also indicates the way to minimize impedance, and that is to manipulate the shape of the cross-section so as to provide more surface area. For its bulk, braided cable is almost pure surface.

Power Supply Distribution and Decoupling

The main consideration for power supply distribution lines is, as for signal lines, to minimize the areas of the current loops. But the power supply lines take on an importance that no signal line has when one considers the fact that these lines have access to every PC board in the system. The very extensiveness of the supply current loops makes it difficult to keep loop areas small. And, a noise glitch on a supply line is a glitch delivered to every board in the system.

The power supply provides low-frequency current to the load, but the inductance of the board-to-board and chip-to-chip distribution network makes it difficult for the power supply to maintain VCC specs on the chip while providing the current spikes that a digital system requires. In addition, the power supply current loop is a very large one, which means there will be a lot of noise pick-up. Figure 19a shows a load circuit trying to draw current spikes from a supply voltage through the line impedance. To the VCC waveform shown in that figure should be added the inductive pick-up associated with a large loop area.

Adding a decoupling capacitor solves two problems: The capacitor acts as a nearby source of charge to supply the current spikes through a smaller line impedance, and it defines a much smaller loop area for the

higher frequency components of EMI. This is illustrated in Figure 19b, which shows the capacitor supplying the current spike, during which VCC drops from 5V by the amount indicated in the figure. Between current spikes the capacitor recovers through the line impedance.

One should resist the temptation to add a resistor or an inductor to the decoupler so as to form a genuine RC or LC low-pass filter because that slows down the speed with which the decoupler cap can be refreshed. Good filtering and good decoupling are not necessarily the same thing.

The current loop for the higher frequency currents, then, is defined by the decoupling cap and the load circuit, rather than by the power supply and the load circuit. For the decoupling cap to be able to provide the current spikes required by the load, the inductance of this current loop must be kept small, which is the same as saying the loop area must be kept small. This is also the requirement for minimizing inductive pick-up in the loop.

There are two kinds of decoupling caps: board decouplers and chip decouplers. A board decoupler will normally be a 10 to 100 μ F electrolytic capacitor placed near to where the power supply enters the PC board, but its placement is relatively non-critical. The purpose of the board decoupler is to refresh the charge on the chip decouplers. The chip decouplers are what actually provide the current spikes to the chips. A chip decoupler will normally be a 0.1 to 1 μ F ceramic capacitor placed near the chip and connected to the chip by traces that minimize the area of the loop formed by the cap and the chip. If a chip decoupler is not properly placed on the board, it will be ineffective as a decoupler

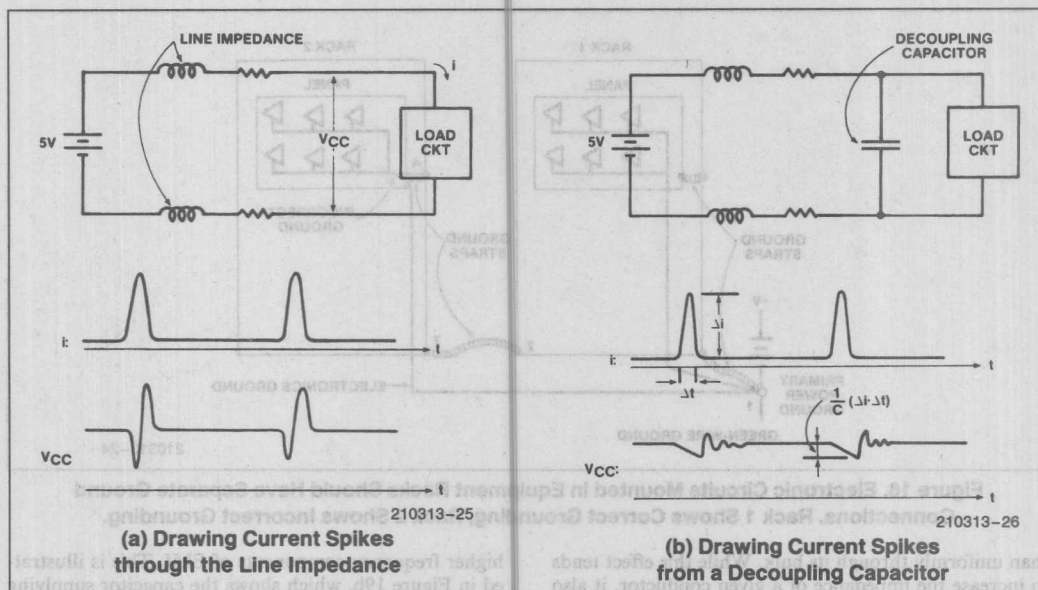


Figure 19. What a Decoupling Capacitor Does

and will serve only to increase the cost of the board. Good and bad placement of decoupling capacitors are illustrated in Figure 20.

Power distribution traces on the PC board need to be laid out so as to obtain minimal area (minimal inductance) in the loops formed by each chip and its decoupler, and by the chip decouplers and the board decoupler. One way to accomplish this goal is to use a power plane. A power plane is the same as a ground plane, but at VCC potential. More economically, a power grid similar to the ground grid previously discussed (Figure 8) can be used. Actually, if the chip decoupling loops are small, other aspects of the power layout are less critical. In other words, power planes and power gridding aren't needed, but power traces *should* be laid in the closest possible proximity to ground traces, prefer-

ably so that each power trace is on the direct opposite side of the board from a ground trace.

Special-purpose power supply distribution buses which mount on the PCB are available. The buses use a parallel flat conductor configuration, one conductor being a VCC line and the other a ground line. Used in conjunction with a gridded ground layout, they not only provide a low-inductance distribution system, but can themselves form part of the ground grid, thus facilitating the PCB layout. The buses are available with and without enhanced bus capacitance, under the names Mini/Bus® and Q/PAC® from Rogers Corp. (5750 E. McKellips, Mesa, AZ 85205).

SELECTING THE VALUE OF THE DECOUPLING CAP

The effectiveness of the decoupling cap has a lot to do with the way the power and ground traces connect this capacitor to the chip. In fact, the area formed by this loop is more important than the value of the capacitance. Then, given that the area of this loop is indeed minimal, it can generally be said that the larger the value of the decoupling cap, the more effective it is, if the cap has a mica, ceramic, glass, or polystyrene dielectric.

It's often said, and not altogether accurately, that the chip decoupler shouldn't have too large a value. There are two reasons for this statement. One is that some capacitors, because of the nature of their dielectrics, tend to become inductive or lossy at higher frequencies. This is true of electrolytic capacitors, but mica, glass,

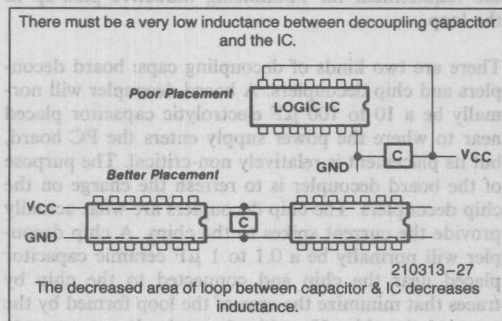


Figure 20. Placement of Decoupling Capacitors

ceramic, and polystyrene dielectrics work well to several hundred MHz. The other reason cited for not using too large a capacitance has to do with lead inductance.

The capacitor with its lead inductance forms a series LC circuit. Below the frequency of series resonance, the net impedance of the combination is capacitive. Above that frequency, the net impedance is inductive. Thus a decoupling capacitor is capacitive only below the frequency of series resonance. The frequency is given by

$$f_0 = \frac{1}{2\pi\sqrt{LC}}$$

where C is the decoupling capacitance and L is the lead inductance between the capacitor and the chip. On a PC board this inductance is determined by the layout, and is the same whether the capacitor dropped into the PCB holes is 0.001 μ F or 1 μ F. Thus, increasing the capacitance lowers the series resonant frequency. In fact, according to the resonant frequency formula, increasing C by a factor of 100 lowers the resonant frequency by a factor of 10.

Figures quoted on the series resonant frequency of a 0.01 μ F capacitor run from 10 to 15 MHz, depending on the lead length. If these numbers were accurate, a 1 μ F capacitor in the same position on the board would have a resonant frequency of 1.0 to 1.5 MHz, and as a decoupler would do more harm than good. However, the numbers are based on a presumed inductance of a given length of wire (the lead length). It should be noted that a "length of wire" has no inductance at all, strictly speaking. Only a complete current loop has inductance, and the inductance depends on the geometry of the loop. Figures quoted on the inductance of a length of wire are based on a presumably "very large" loop area, such that the magnetic field produced by the return current has no cancellation effect on the field produced by the current in the given length of wire. Such a loop geometry is not and should not be the case with the decoupling loop.

Figure 21 shows VCC waveforms, measured between pins 40 and 20 (VCC and VSS) of an 8751 CPU, for several conditions of decoupling on a PC board that has a decoupling loop area slightly larger than necessary. These photographs show the effects of increasing the decoupling capacitance and decreasing the area of the decoupling loop. The indications are that a 1 μ F capacitor is better than a 0.1 μ F capacitor, which in turn is better than nothing, and that the board should have been laid out with more attention paid to the area of the decoupling loop.

Figure 21e was obtained using a special-purpose experimental capacitor designed by Rogers Corp. (Q-Pac Division, Mesa, AZ) for use as a decoupler. It consists of two parallel plates, the length of a 40-pin DIP, separated by a ceramic dielectric. Sandwiched between the

CPU chip and the PCB (or between the CPU socket and the PCB), it makes connection to pins 40 and 20, forming a leadless decoupling capacitor. It is obviously a configuration of minimal inductance. Unfortunately, the particular sample tested had only 0.07 μ F of capacitance and so was unable to prevent the 1 MHz ripple as effectively as the configuration of Figure 21d. It seems apparent, though, that with more capacitance this part will alleviate a lot of decoupling problems.

THE CASE FOR ON-BOARD VOLTAGE REGULATION

To complicate matters, supply line glitches aren't always picked up in the distribution networks, but can come from the power supply circuit itself. In that case, a well-designed distribution network faithfully delivers the glitch throughout the system. The VCC glitch in Figure 22 was found to be coming from within a bench power supply in response to the EMP produced by an induction coil spark generator that was being used at Intel during a study of noise sensitivity. The VCC glitch is about 400 mV high and some 20 μ s in duration. Normal board decoupling techniques were ineffective in removing it, but adding an on-board voltage regulator chip did the job.

Thus, a good case can be made in favor of using a voltage regulator chip on each PCB, instead of doing all the voltage regulation at the supply circuit. This eases requirements on the heat-sinking at the supply circuit, and alleviates much of the distribution and board decoupling headaches. However, it also brings in the possibility that different boards would be operating at slightly different VCC levels due to tolerance in the regulator chips; this then leads to slightly different logic levels from board to board. The implications of that may vary from nothing to latch-up, depending on what kinds of chips are on the boards, and how they react to an input "high" that is perhaps 0.4V higher than local VCC.

Recovering Gracefully from a Software Upset

Even when one follows all the best guidelines for designing for a noisy environment, it's always possible for a noise transient to occur which exceeds the circuit's immunity level. In that case, one can strive at least for a graceful recovery.

Graceful recovery schemes involve additional hardware and/or software which is supposed to return the system to a normal operating mode after a software upset has occurred. Two decisions have to be made: How to recognize when an upset has occurred, and what to do about it.

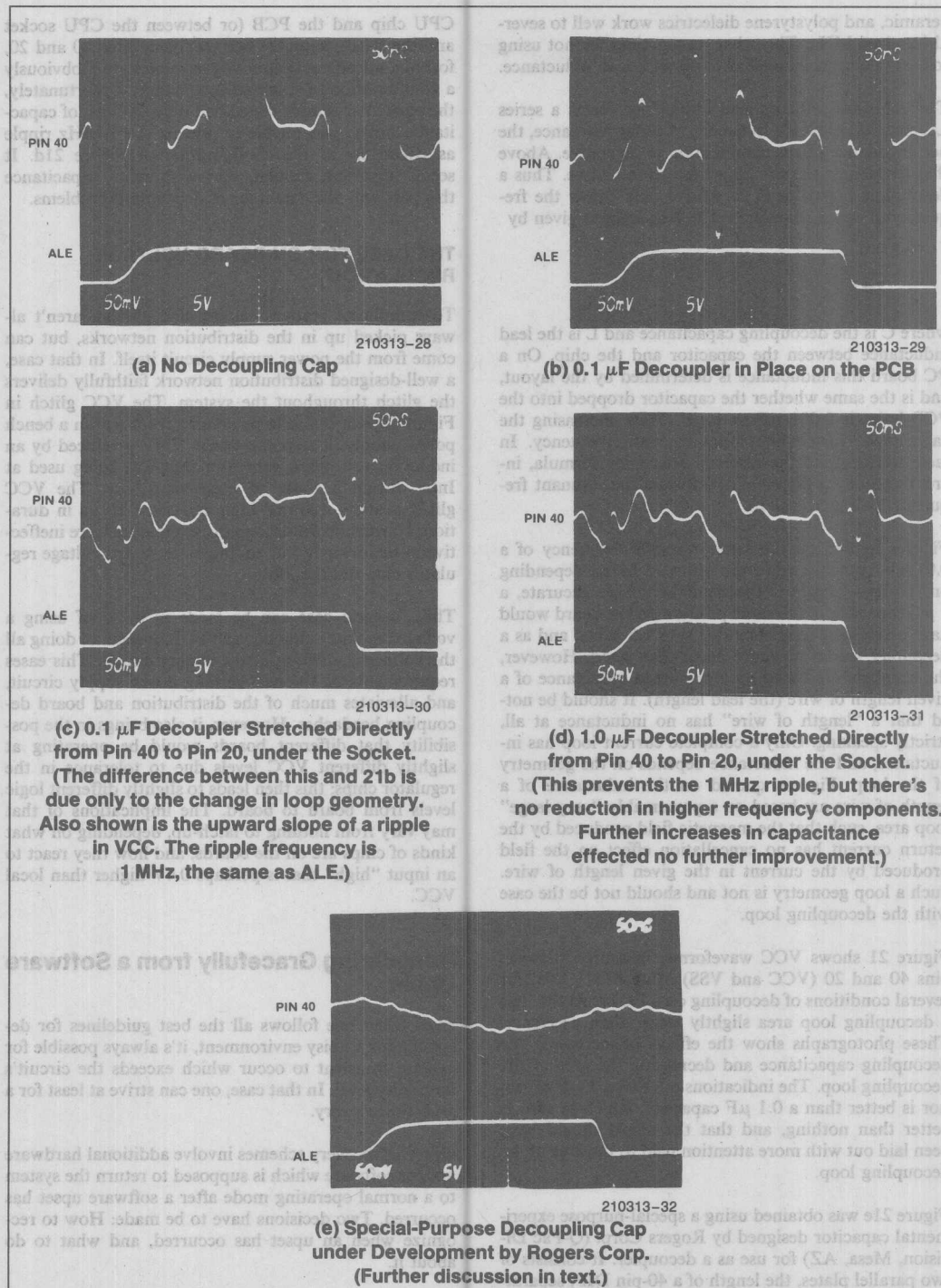


Figure 21. Noise on VCC Line

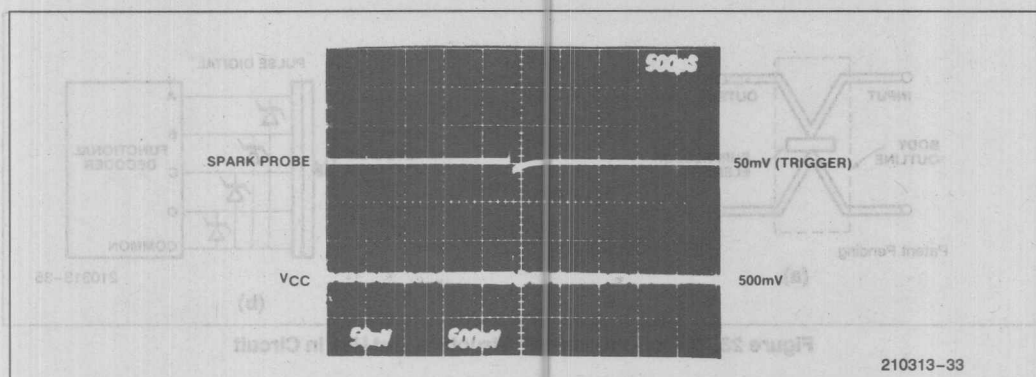


Figure 22. EMP-Induced Glitch

If the designer knows what kinds and combinations of outputs can legally be generated by the system, he can use gates to recognize and flag the occurrence of an illegal state of affairs. The flag can then trigger a jump to a recovery routine which then may check or re-initialize data, perhaps output an error message, or generate a simple reset.

The most reliable scheme is to use a so-called watchdog circuit. Here the CPU is programmed to generate a periodic signal as long as the system is executing instructions in an expected manner. The periodic signal is then used to hold off a circuit that will trigger a jump to a recovery routine. The periodic signal needs to be AC-coupled to the trigger circuit so that a "stuck-at" fault won't continue to hold off the trigger. Then, if the processor locks up someplace, the periodic signal is lost and the watchdog triggers a reset.

In practice, it may be convenient to drive the watchdog circuit with a signal which is being generated anyway by the system. One needs to be careful, however, that an upset does in fact discontinue that signal. Specifically, for example, one could use one of the digit drive signals going to a multiplexed display. But display scanning is often handled in response to a timer-interrupt, which may continue operating even though the main program is in a failure mode. Even so, with a little extra software, the signal can be used to control the watchdog (see Reference 8 on this).

Simpler schemes can work well for simpler systems. For example, if a CPU isn't doing anything but scanning and decoding a keyboard, there's little to lose and much to gain by simply resetting it periodically with an astable multivibrator. It only takes about 13 μ s (at 6 MHz) to reset an 8048 if the clock oscillator is already running.

A zero-cost measure is simply to fill all unused program memory with NOPs and JMPs to a recovery routine. The effectiveness of this method is increased by writing the program in segments that are separated by

NOPs and JMPs. It's still possible, of course, to get hung up in a data table or something. But you get a lot of protection, for the cost.

Further discussion of graceful recovery schemes can be found in Reference 13.

Special Problem Areas

ESD

MOS chips have some built-in protection against a static charge build-up on the pins, as would occur during normal handling, but there's no protection against the kinds of current levels and rise times that occur in a genuine electrostatic spark. These kinds of discharges can blow a crater in the silicon.

It must be recognized that connecting CPU pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges makes an extremely fragile configuration. Buffering them is the very least one can do. But buffering doesn't completely solve the problem, because then the buffer chips will sustain the damage (even TTL); therefore, one might consider mounting the buffer chips in sockets for ease of replacement.

Transient suppressors, such as the TranZorbs® made by General Semiconductor Industries (Tempe, AZ), may in the long run provide the cheapest protection if their "zero inductance" structure is used. The structure and circuit application are shown in Figure 23.

The suppressor element is a pn junction that operates like a Zener diode. Back-to-back units are available for AC operation. The element is more or less an open circuit at normal system voltage (the standoff voltage rating for the device), and conducts like a Zener diode at the clamping voltage.

The lead inductance in the conventional transient suppressor package makes the conventional package essen-

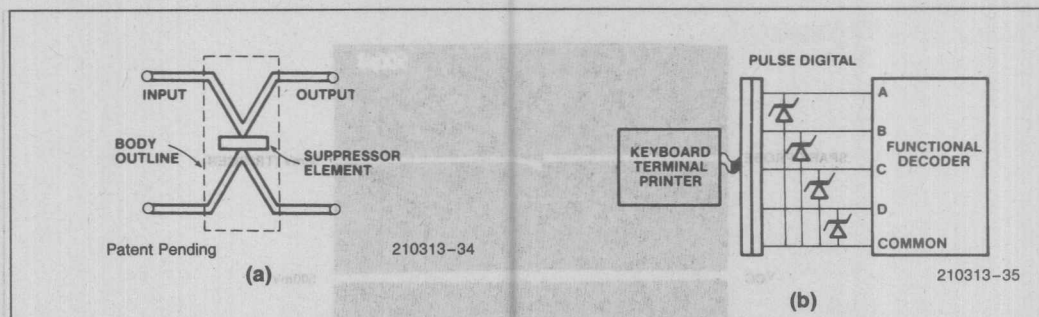


Figure 23. "Zero-Inductance" Structure and Use in Circuit

tially useless for protection against ESD pulses, owing to the fast rise of these pulses. The "zero inductance" units are available singly in a 4-pin DIP, and in arrays of four to a 16-pin DIP for PCB level protection. In that application they should be mounted in close proximity to the chips they protect.

In addition, metal enclosures or frames or parts that can receive an ESD spark should be connected by braided cable to the green-wire ground. Because of the ground impedance, ESD current shouldn't be allowed to flow through any signal ground, even if the chips are protected by transient suppressors. A 35 kV ESD spark can always spare a few hundred volts to drive a fast current pulse down a signal ground line if it can't find a braided cable to follow. Think how delighted your 8048 will be to find its VSS pin 250V higher than VCC for a few 10s of nanoseconds.

THE AUTOMOTIVE ENVIRONMENT

The automobile presents an extremely hostile environment for electronic systems. There are several parts to it:

1. Temperature extremes from -40°C to $+125^{\circ}\text{C}$ (under the hood) or $+85^{\circ}\text{C}$ (in the passenger compartment)
2. Electromagnetic pulses from the ignition system
3. Supply line transients that will knock your socks off

One needs to take a long, careful look at the temperature extremes. The allowable storage temperature range for most Intel MOS chips is -65°C to $+150^{\circ}\text{C}$, although some chips have a maximum storage temperature rating of $+125^{\circ}\text{C}$. In operation (or "under bias," as the data sheets say) the allowable ambient temperature range depends on the product grade, as follows:

Grade	Ambient Temperature	
	Min	Max
Commercial	0	70
Industrial	-40	$+85$
Automotive	-40	$+110$
Military	-55	$+125$

The different product grades are actually the same chip, but tested according to different standards. Thus, a given commercial-grade chip might actually pass military temperature requirements, but not have been tested for it. (Of course, there are other differences in grading requirements having to do with packaging, burn-in, traceability, etc.)

In any case, it's apparent that commercial-grade chips can't be used safely in automotive applications, not even in the passenger compartment. Industrial-grade chips can be used in the passenger compartment, and automotive or military chips are required in under-the-hood applications.

Ignition noise, CB radios, and that sort of thing are probably the least of your worries. In a poorly designed system, or in one that has not been adequately tested for the automotive environment, this type of EMI might cause a few software upsets, but not destroy chips.

The major problem, and the one that seems to come as the biggest surprise to most people, is the line transients. Regrettably, the 12V battery is not actually the source of power when the car is running. The charging system is, and it's not very clean. The only time the battery is the real source of power is when the car is first being started, and in that condition the battery terminals may be delivering about 5V or 6V. As follows is a brief description of the major idiosyncracies of the "12V" automotive power line.

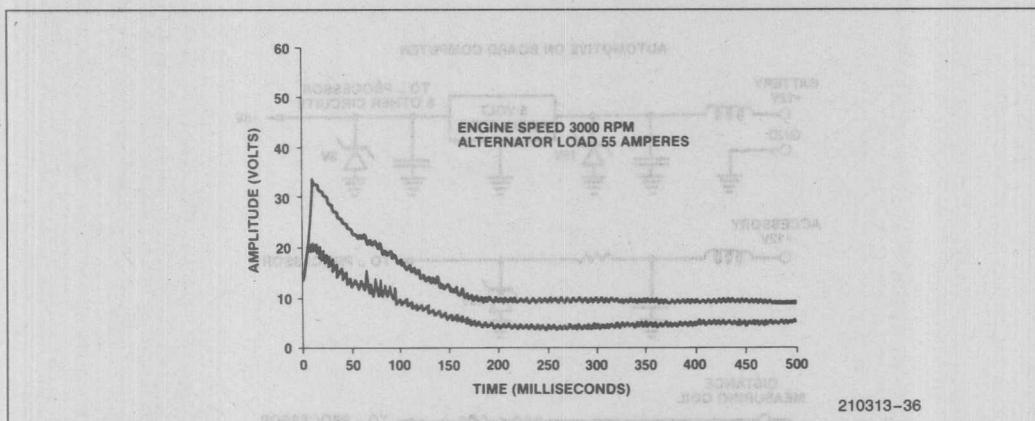


Figure 24. Typical Load Dump Transients

- An abrupt reduction in the alternator load causes a positive voltage transient called "load dump." In a load dump transient the line voltage rises to 20V or 30V in a few μ s, then decays exponentially with a time constant of about 100 μ s, as shown in Figure 24. Much higher peak voltages and longer decay times have also been reported. The worst case load dump is caused by disconnecting a low battery from the alternator circuit while the alternator is running. Normally this would happen intermittently when the battery terminal connections are defective.
- When the ignition is turned off, as the field excitation decays, the line voltage can go to between -40V and -100V for 100 μ s or more.
- Miscellaneous solenoid switching transients, such as the one shown in Figure 25, can drive the line to + or -200V to 400V for several μ s.
- Mutual coupling between unshielded wires in long harnesses can induce 100V and 200V transients in unprotected circuits.

What all this adds up to is that people in the business of building systems for automotive applications need a comprehensive testing program. An SAE guideline which describes the automotive environment is available to designers: SAE J1211, "Recommended Environmental Practices for Electronic Equipment Design," 1980 SAE Handbook, Part 1, pp. 22.80-22.96.

Some suggestions for protecting circuitry are shown in Figure 26. A transient suppressor is placed in front of the regulator chip to protect it. Since the rise times in these transients are not like those in ESD pulses, lead inductance is less critical and conventional devices can be used. The regulator itself is pretty much of a necessity, since a load dump transient is simply not going to be removed by any conventional LC or RC filter.

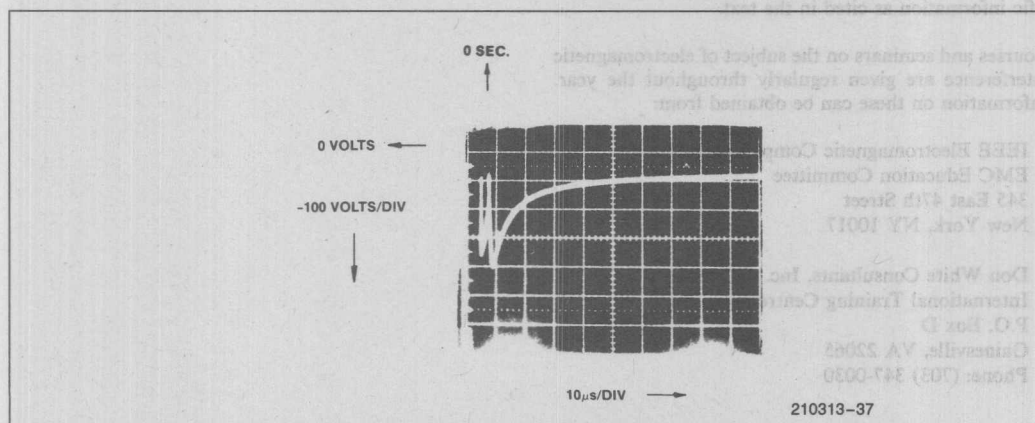


Figure 25. Transient Created by De-energizing an Air Conditioning Clutch Solenoid

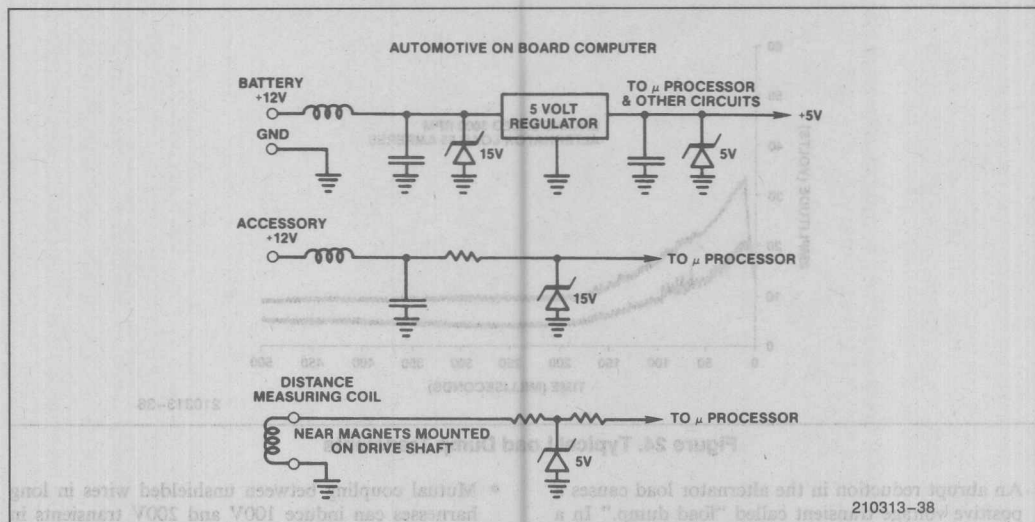


Figure 26. Use of Transient Suppressors in Automotive Applications

Special I/O interfacing is also required, because of the need for high tolerance to voltage transients, input noise, input/output isolation, etc. In addition, switches that are being monitored or driven by these buffers are usually referenced to chassis ground instead of signal ground, and in a car there can be many volts difference between the two. I/O interfacing is discussed in Reference 2.

Parting Thoughts

The main sources of information for this Application Note were the references by Ott and by White. Reference 5 is probably the finest treatment currently available on the subject. The other references provided specific information as cited in the text.

Courses and seminars on the subject of electromagnetic interference are given regularly throughout the year. Information on these can be obtained from:

IEEE Electromagnetic Compatibility Society
EMC Education Committee
345 East 47th Street
New York, NY 10017

Don White Consultants, Inc.
International Training Centre
P.O. Box D
Gainesville, VA 22065
Phone: (703) 347-0030

The EMC Education committee has available a video tape: "Introduction to EMC—A Video Training Tape," by Henry Ott. Don White Consultants offers a series of training courses on many different aspects of electromagnetic compatibility. Most organizations that sponsor EMC courses also offer in-plant presentations.

REFERENCES

1. Clark, O.M., "Electrostatic Discharge Protection Using Silicon Transient Suppressors," *Proceedings of the Electrical Overstress/Electrostatic Discharge Symposium*. Reliability Analysis Center, Rome Air Development Center, 1979.
2. Kearney, M; Shreve, J.; and Vincent, W., "Microprocessor Based Systems in the Automobile: Custom Integrated Circuits Provide an Effective Interface," *Electronic Engine Management and Driveline Control Systems*, SAE Publication SP-481, 810160, pp. 93-102.
3. King, W.M. and Reynolds, D., "Personnel Electrostatic Discharge: Impulse Waveforms Resulting From ESD of Humans Directly and Through Small Hand-Held Metallic Objects Intervening in the Discharge Path," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility*, pp. 577-590, Aug. 1981.
4. Ott, H., "Digital Circuit Grounding and Interconnection," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility*, pp. 292-297, Aug. 1981.
5. Ott, H., *Noise Reduction Techniques in Electronic Systems*. New York: Wiley, 1976.
6. *1981 Interference Technology Engineers' Master (ITEM) Directory and Design Guide*. R. and B. Enterprises, P.O. Box 328, Plymouth Meeting, PA 19426.
7. SAE J1211, "Recommended Environmental Practices for Electronic Equipment Design," *1980 SAE Handbook*, Part 1, pp. 22.80-22.96.
8. Smith, L., "A Watchdog Circuit for Microcomputer Based Systems," *Digital Design*, pp. 78, 79, Nov. 1979.
9. *TranZorb Quick Reference Guide*. General Semiconductor Industries, P.O. Box 3078, Tempe, AZ 85281.
10. Tucker, T.J., "Spark Initiation Requirements of a Secondary Explosive," *Annals of the New York Academy of Sciences*, Vol 152, Article I, pp. 643-653, 1968.
11. White, D., *Electromagnetic Interference and Compatibility, Vol. 3: EMI Control Methods and Techniques*. Don White Consultants, 1973.
12. White, D., *EMI Control in the Design of Printed Circuit Boards and Backplanes*. Don White Consultants, 1981.
13. Yarkoni, B. and Wharton, J., "Designing Reliable Software for Automotive Applications," *SAE Transactions*, 790237, July 1979.

Oscillators for Microcontrollers

TOM WILLIAMSON
MICROCONTROLLER
TECHNICAL MARKETING

Order Number: 230659-001

OSCILLATORS FOR MICROCONTROLLERS

PAGE

9-15

Troubleshooting Oscillator Problems 9-16

APPENDIX A: QUARTZ AND CERAMIC

RESONATOR FORMULAS 9-18

APPENDIX B: OSCILLATOR ANALYSIS

PROGRAM 9-20

CONTENTS

PAGE

INTRODUCTION 9-27

FEEDBACK OSCILLATORS 9-27

Loop Gain 9-27

How Feedback Oscillators Work 9-28

The Positive Reactance Oscillator 9-28

QUARTZ CRYSTALS 9-29

Crystal Parameters 9-29

Equivalent Circuit 9-29

Load Capacitance 9-30

"Series" vs. "Parallel" Crystals 9-30

Equivalent Series Resistance 9-30

Frequency Tolerance 9-31

Drive Level 9-31

CERAMIC RESONATORS 9-31

Specifications for Ceramic Resonators . . . 9-32

OSCILLATOR DESIGN

CONSIDERATIONS 9-32

On-Chip Oscillators 9-32

Crystal Specifications 9-32

Oscillation Frequency 9-33

Selection of CX1 and CX2 9-33

Placement of Components 9-33

Clocking Other Chips 9-33

External Oscillators 9-34

Gate Oscillators vs. Discrete Devices . . . 9-36

Fundamental vs. Overtone

Operation 9-37

"Series" vs. "Parallel" Operation 9-37

CONTENTS

MORE ABOUT USING THE "ON-CHIP" OSCILLATORS

Oscillator Calculations	9-37
Start-Up Characteristics	9-39
Steady-State Characteristics	9-41
Pin Capacitance	9-42
MCS®-51 Oscillator	9-42
MCS®-48 Oscillator	9-42

Equivalent Circuit	9-43
Load Capacitance	9-43
"Series" vs. "Parallel" Crystals	9-43
Equivalent Series Resistance	9-43
Frequency Tolerance	9-43
Drive Level	9-43
CERAMIC RESONATORS	9-43
Specifications for Ceramic Resonators	9-43

OSCILLATOR DESIGN CONSIDERATIONS	9-43
On-Chip Oscillators	9-43
Crystal Specifications	9-43
Oscillation Frequency	9-43
Selection of CXT and CXTS	9-43
Placement of Components	9-43
Clocking Other Chips	9-43
External Oscillators	9-43
Gate Oscillators vs. Discrete Devices	9-43
Fundamental vs. Overtone Operation	9-43
"Series" vs. "Parallel" Operation	9-43

CONTENTS

PAGE

Pre-Production Tests	9-45
Troubleshooting Oscillator Problems	9-46

APPENDIX A: QUARTZ AND CERAMIC RESONATOR FORMULAS

9-48

APPENDIX B: OSCILLATOR ANALYSIS PROGRAM

9-50

INTRODUCTION

Intel's microcontroller families (MCS®-48, MCS®-51, and iACX-96) contain a circuit that is commonly referred to as the "on-chip oscillator". The on-chip circuitry is not itself an oscillator, of course, but an amplifier that is suitable for use as the amplifier part of a feedback oscillator. The data sheets and Microcontroller Handbook show how the on-chip amplifier and several off-chip components can be used to design a working oscillator. With proper selection of off-chip components, these oscillator circuits will perform better than almost any other type of clock oscillator, and by almost any criterion of excellence. The suggested circuits are simple, economical, stable, and reliable.

We offer assistance to our customers in selecting suitable off-chip components to work with the on-chip oscillator circuitry. It should be noted, however, that Intel cannot assume the responsibility of writing specifications for the off-chip components of the complete oscillator circuit, nor of guaranteeing the performance of the finished design in production, anymore than a transistor manufacturer, whose data sheets show a number of suggested amplifier circuits, can assume responsibility for the operation, in production, of any of them.

We are often asked why we don't publish a list of required crystal or ceramic resonator specifications, and recommend values for the other off-chip components. This has been done in the past, but sometimes with consequences that were not intended.

Suppose we suggest a maximum crystal resistance of 30 ohms for some given frequency. Then your crystal supplier tells you the 30-ohm crystals are going to cost twice as much as 50-ohm crystals. Fearing that Intel will not "guarantee operation" with 50-ohm crystals, you order the expensive ones. In fact, Intel guarantees only what is embodied within an Intel product. Besides, there is no reason why 50-ohm crystals couldn't be used, if the other off-chip components are suitably adjusted.

Should we recommend values for the other off-chip components? Should we do it for 50-ohm crystals or 30-ohm crystals? With respect to what should we optimize their selection? Should we minimize start-up time or maximize frequency stability? In many applications, neither start-up time nor frequency stability are particularly critical, and our "recommendations" are only restricting your system to unnecessary tolerances. It all depends on the application.

Although we will neither "specify" nor "recommend" specific off-chip components, we do offer assistance in these tasks. Intel application engineers are available to provide whatever technical assistance may be needed or desired by our customers in designing with Intel products.

This Application Note is intended to provide such assistance in the design of oscillator circuits for microcontroller systems. Its purpose is to describe in a practical manner how oscillators work, how crystals and ceramic resonators work (and thus how to spec them), and what the on-chip amplifier looks like electronically and what its operating characteristics are. A BASIC program is provided in Appendix II to assist the designer in determining the effects of changing individual parameters. Suggestions are provided for establishing a pre-production test program.

FEEDBACK OSCILLATORS

Loop Gain

Figure 1 shows an amplifier whose output line goes into some passive network. If the input signal to the amplifier is v_1 , then the output signal from the amplifier is $v_2 = Av_1$ and the output signal from the passive network is $v_3 = \beta v_2 = \beta Av_1$. Thus βA is the overall gain from terminal 1 to terminal 3.

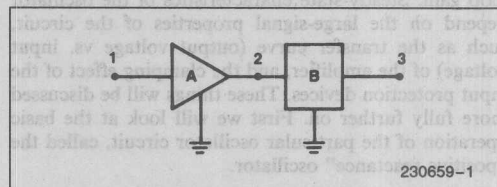


Figure 1. Factors in Loop Gain

Now connect terminal 1 to terminal 3, so that the signal path forms a loop: 1 to 2 to 3, which is also 1. Now we have a feedback loop, and the gain factor βA is called the *loop gain*.

Gain factors are complex numbers. That means they have a magnitude and a phase angle, both of which vary with frequency. When writing a complex number, one must specify both quantities, magnitude and angle. A number whose magnitude is 3, and whose angle is 45 degrees is commonly written this way: $3\angle 45^\circ$. The number 1 is, in complex number notation, $1\angle 0^\circ$, while -1 is $1\angle 180^\circ$.

By closing the feedback loop in Figure 1, we force the equality

$$v_1 = \beta Av_1$$

This equation has two solutions:

- 1) $v_1 = 0$;
- 2) $\beta A = 1\angle 0^\circ$.

in effect. In the first solution the circuit is quiescent (no output signal). If you're trying to make an oscillator, a no-signal condition is unacceptable. There are ways to guarantee that the second solution is the one that will be in effect, and that the quiescent condition will be excluded.

How Feedback Oscillators Work

A feedback oscillator amplifies its own noise and feeds it back to itself in exactly the right phase, at the oscillation frequency, to build up and reinforce the desired oscillations. Its ability to do that depends on its loop gain. First, oscillations can occur only at the frequency for which the loop gain has a phase angle of 0 degrees. Second build-up of oscillations will occur only if the loop gain exceeds 1 at the frequency. Build-up continues until nonlinearities in the circuit reduce the average value of the loop gain to exactly 1.

Start-up characteristics depend on the small-signal properties of the circuit, specifically, the small-signal loop gain. Steady-state characteristics of the oscillator depend on the large-signal properties of the circuit, such as the transfer curve (output voltage vs. input voltage) of the amplifier, and the clamping effect of the input protection devices. These things will be discussed more fully further on. First we will look at the basic operation of the particular oscillator circuit, called the "positive reactance" oscillator.

The Positive Reactance Oscillator

Figure 2 shows the configuration of the positive reactance oscillator. The inverting amplifier, working into the impedance of the feedback network, produces an output signal that is nominally 180 degrees out of phase with its input. The feedback network must provide an additional 180 degrees phase shift, such that the overall loop gain has zero (or 360) degrees phase shift at the oscillation frequency.

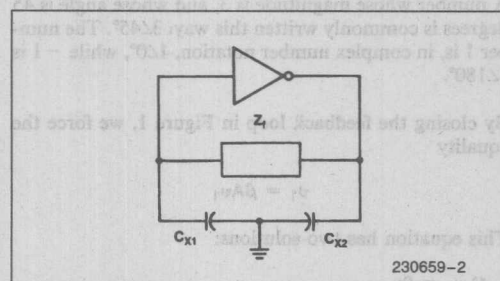


Figure 2. Positive Reactance Oscillator

necessary that the feedback element Z_f have a positive reactance. That is, it must be inductive. Then, the frequency at which the phase angle is zero is approximately the frequency at which

$$X_f = \frac{+1}{\omega C}$$

where X_f is the reactance of Z_f (the total Z_f being $R_f + jX_f$, and C is the series combination of C_{X1} and C_{X2}).

$$C = \frac{C_{X1} C_{X2}}{C_{X1} + C_{X2}}$$

In other words, Z_f and C form a parallel resonant circuit.

If Z_f is an inductor, then $X_f = \omega L$, and the frequency at which the loop gain has zero phase is the frequency at which

$$\omega L = \frac{1}{\omega C}$$

or

$$\omega = \frac{1}{\sqrt{LC}}$$

Normally, Z_f is not an inductor, but it must still have a positive reactance in order for the circuit to oscillate. There are some piezoelectric devices on the market that show a positive reactance, and provide a more stable oscillation frequency than an inductor will. Quartz crystals can be used where the oscillation frequency is critical, and lower cost ceramic resonators can be used where the frequency is less critical.

When the feedback element is a piezoelectric device, this circuit configuration is called a Pierce oscillator. The advantage of piezoelectric resonators lies in their property of providing a wide range of positive reactance values over a very narrow range of frequencies. The reactance will equal $1/\omega C$ at some frequency within this range, so the oscillation frequency will be within the same range. Typically, the width of this range is

only 0.3% of the nominal frequency of a quartz crystal, and about 3% of the nominal frequency of a ceramic resonator. With relatively little design effort, frequency accuracies of 0.03% or better can be obtained with quartz crystals, and 0.3% or better with ceramic resonators.

QUARTZ CRYSTALS

The crystal resonator is a thin slice of quartz sandwiched between two electrodes. Electrically, the device looks pretty much like a 5 or 6 pF capacitor, except that over certain ranges of frequencies the crystal has a positive (i.e., inductive) reactance.

The ranges of positive reactance originate in the piezoelectric property of quartz: Squeezing the crystal generates an internal E-field. The effect is reversible: Applying an AC E-field causes the crystal to vibrate. At certain vibrational frequencies there is a mechanical resonance. As the E-field frequency approaches a frequency of mechanical resonance, the measured reactance of the crystal becomes positive, as shown in Figure 3.

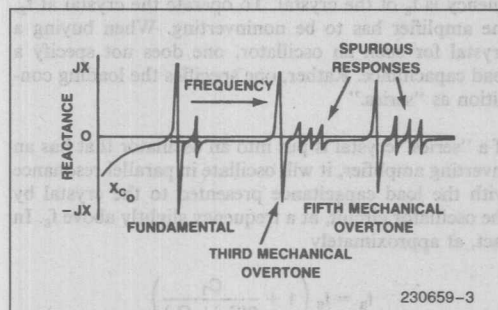


Figure 3. Crystal Reactance vs. Frequency

Typically there are several ranges of frequencies where in the reactance of the crystal is positive. Each range corresponds to a different mode of vibration in the crystal. The main resonances are the so-called fundamental response and the third and fifth overtone responses.

The overtone responses shouldn't be confused with the harmonics of the fundamental. They're not harmonics, but different vibrational modes. They're not in general at exact integer multiples of the fundamental frequency. There will also be "spurious" responses, occurring typically a few hundred KHz above each main response.

To assure that an oscillator starts in the desired mode on power-up, something must be done to suppress the loop gain in the undesired frequency ranges. The crystal itself provides some protection against unwanted modes of oscillation; too much resistance in that mode, for example. Additionally, junction capacitances in the amplifying devices tend to reduce the gain at higher frequencies, and thus may discriminate against unwanted modes. In some cases a circuit fix is necessary, such as inserting a trap, a phase shifter, or ferrite beads to kill oscillations in unwanted modes.

Crystal Parameters

Equivalent Circuit

Figure 4 shows an equivalent circuit that is used to represent the crystal for circuit analysis.

The R_1 - L_1 - C_1 branch is called the motivational arm of the crystal. The values of these parameters derive from the mechanical properties of the crystal and are constant for a given mode of vibration. Typical values for various nominal frequencies are shown in Table 1.

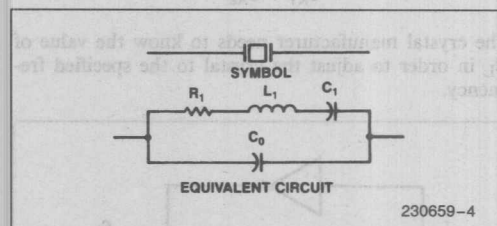


Figure 4. Quartz Crystal: Symbol and Equivalent Circuit

C_0 is called the shunt capacitance of the crystal. This is the capacitance of the crystal's electrodes and the mechanical holder. If one were to measure the reactance of the crystal at a frequency far removed from a resonance frequency, it is the reactance of this capacitance that would be measured. It's normally 3 to 7 pF.

Table 1. Typical Crystal Parameters

Frequency MHz	R_1 ohms	L_1 mH	C_1 pF	C_0 pF
2	100	520	0.012	4
4.608	36	117	0.010	2.9
11.25	19	8.38	0.024	5.4

The series resonant frequency of the crystal is the frequency at which L_1 and C_1 are in resonance. This frequency is given by

$$f_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

At this frequency the impedance of the crystal is R_1 in parallel with the reactance of C_0 . For most purposes, this impedance is taken to be just R_1 , since the reactance of C_0 is so much larger than R_1 .

Load Capacitance

A crystal oscillator circuit such as the one shown in Figure 2 (redrawn in Figure 5) operates at the frequency for which the crystal is antiresonant (ie, parallel-resonant) with the total capacitance across the crystal terminals external to the crystal. This total capacitance external to the crystal is called the load capacitance.

As shown in Figure 5, the load capacitance is given by

$$C_L = \frac{C_{X1} C_{X2}}{C_{X1} + C_{X2}} + C_{stray}$$

The crystal manufacturer needs to know the value of C_L in order to adjust the crystal to the specified frequency.

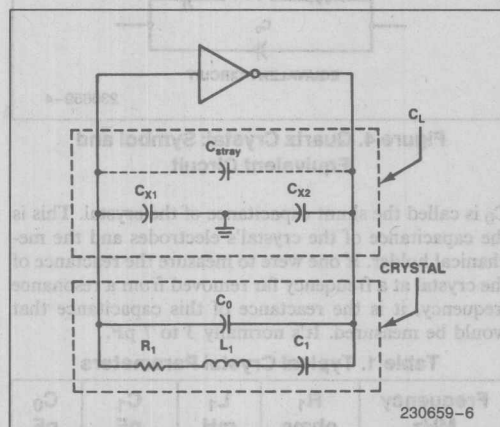


Figure 5. Load Capacitance

The adjustment involves putting the crystal in series with the specified C_L , and then "trimming" the crystal to obtain resonance of the series combination of the crystal and C_L at the specified frequency. Because of the high Q of the crystal, the resonant frequency of the series combination of the crystal and C_L is the same as

the antiresonant frequency of the parallel combination of the crystal and C_L . This frequency is given by

$$f_a = \frac{1}{2\pi\sqrt{L_1 C_1 (C_L + C_0) / (C_1 + C_L + C_0)}}$$

These frequency formulas are derived (in Appendix A) from the equivalent circuit of the crystal, using the assumptions that the Q of the crystal is extremely high, and that the circuit external to the crystal has no effect on the frequency other than to provide the load capacitance C_L . The latter assumption is not precisely true, but it is close enough for present purposes.

"Series" vs. "Parallel" Crystals

There is no such thing as a "series cut" crystal as opposed to a "parallel cut" crystal. There are different cuts of crystal, having to do with the parameters of its motional arm in various frequency ranges, but there is no special cut for series or parallel operation.

An oscillator is series resonant if the oscillation frequency is f_s of the crystal. To operate the crystal at f_s , the amplifier has to be noninverting. When buying a crystal for such an oscillator, one does not specify a load capacitance. Rather, one specifies the loading condition as "series."

If a "series" crystal is put into an oscillator that has an inverting amplifier, it will oscillate in parallel resonance with the load capacitance presented to the crystal by the oscillator circuit, at a frequency slightly above f_s . In fact, at approximately

$$f_a = f_s \left(1 + \frac{C_1}{2(C_L + C_0)} \right)$$

This frequency would typically be about 0.02% above f_s .

Equivalent Series Resistance

The "series resistance" often listed on quartz crystal data sheets is the real part of the crystal impedance at the crystal's calibration frequency. This will be R_1 if the calibration frequency is the series resonant frequency of the crystal. If the crystal is calibrated for parallel resonance with a load capacitance C_L , the equivalent series resistance will be

$$ESR = R_1 \left(1 + \frac{C_0}{C_L} \right)^2$$

The crystal manufacturer measures this resistance at the calibration frequency during the same operation in which the crystal is adjusted to the calibration frequency.

Frequency Tolerance

Frequency tolerance as discussed here is not a requirement on the crystal, but on the complete oscillator. There are two types of frequency tolerances on oscillators: frequency *accuracy* and frequency *stability*. Frequency accuracy refers to the oscillator's ability to run at an exact specified frequency. Frequency stability refers to the constancy of the oscillation frequency.

Frequency accuracy requires mainly that the oscillator circuit present to the crystal the same load capacitance that it was adjusted for. Frequency stability requires mainly that the load capacitance be constant.

In most digital applications the accuracy and stability requirements on the oscillator are so wide that it makes very little difference what load capacitance the crystal was adjusted to, or what load capacitance the circuit actually presents to the crystal. For example, if a crystal was calibrated to a load capacitance of 25 pF, and is used in a circuit whose actual load capacitance is 50 pF, the frequency error on that account would be less than 0.01%.

In a positive reactance oscillator, the crystal only needs to be in the intended response mode for the oscillator to satisfy a 0.5% or better frequency tolerance. That's because for any load capacitance the oscillation frequency is certain to be between the crystal's resonant and anti-resonant frequencies.

Phase shifts that take place within the amplifier part of the oscillator will also affect frequency accuracy and stability. These phase shifts can normally be modeled as an "output capacitance" that, in the positive reactance oscillator, parallels C_{X2} . The predictability and constancy of this output capacitance over temperature and device sample will be the limiting factor in determining the tolerances that the circuit is capable of holding.

Drive Level

Drive level refers to the power dissipation in the crystal. There are two reasons for specifying it. One is that the parameters in the equivalent circuit are somewhat dependent on the drive level at which the crystal is calibrated. The other is that if the application circuit exceeds the test drive level by too much, the crystal may be damaged. Note that the terms "test drive level" and "rated drive level" both refer to the drive level at which the crystal is calibrated. Normally, in a microcontroller system, neither the frequency tolerances nor the power levels justify much concern for this specification. Some crystal manufacturers don't even require it for microprocessor crystals.

In a positive reactance oscillator, if one assumes the peak voltage across the crystal to be something in the neighborhood of V_{CC} , the power dissipation can be approximated as

$$P = 2R_1 [\pi f (C_L + C_0) V_{CC}]^2$$

This formula is derived in Appendix A. In a 5V system, P rarely evaluates to more than a milliwatt. Crystals with a standard 1 or 2 mW drive level rating can be used in most digital systems.

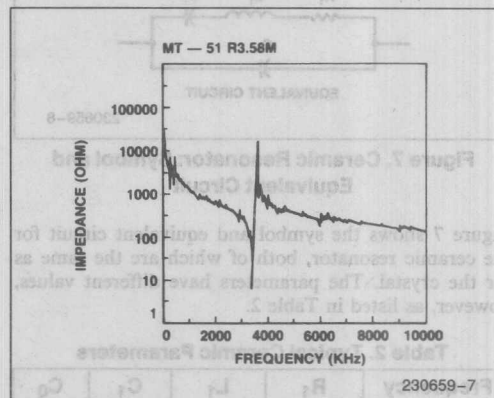


Figure 6. Ceramic Resonator Impedance vs. Frequency (Test Data Supplied by NTK Technical Ceramics)

CERAMIC RESONATORS

Ceramic resonators operate on the same basic principles as a quartz crystal. Like quartz crystals, they are piezoelectric, have a reactance versus frequency curve similar to a crystal's, and an equivalent circuit that looks just like a crystal's (with different parameter values, however).

The frequency tolerance of a ceramic resonator is about two orders of magnitude wider than a crystal's, but the ceramic is somewhat cheaper than a crystal. It may be noted for comparison that quartz crystals with relaxed tolerances cost about twice as much as ceramic resonators. For purposes of clocking a microcontroller, the frequency tolerance is often relatively noncritical, and the economic consideration becomes the dominant factor.

Figure 6 shows a graph of impedance magnitude versus frequency for a 3.58 MHz ceramic resonator. (Note that Figure 6 is a graph of $|Z_f|$ versus frequency, where

as Figure 5 is a graph of X_f versus frequency.) A number of spurious responses are apparent in Figure 6. The manufacturers state that spurious responses are more prevalent in the lower frequency resonators (kHz range) than in the higher frequency units (MHz range). For our purposes only the MHz range ceramics need to be considered.

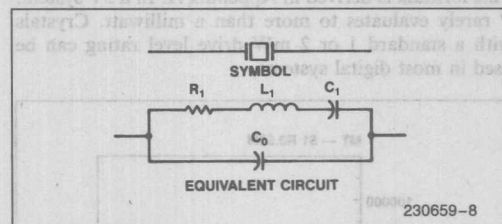


Figure 7. Ceramic Resonator: Symbol and Equivalent Circuit

Figure 7 shows the symbol and equivalent circuit for the ceramic resonator, both of which are the same as for the crystal. The parameters have different values, however, as listed in Table 2.

Table 2. Typical Ceramic Parameters

Frequency MHz	R_1 ohms	L_1 mH	C_1 pF	C_0 pF
3.58	7	0.113	19.6	140
6.0	8	0.094	8.3	60
8.0	7	0.092	4.6	40
11.0	10	0.057	3.9	30

Note that the motional arm of the ceramic resonator tends to have less resistance than the quartz crystal and also a vastly reduced L_1/C_1 ratio. This results in the motional arm having a Q (given by $(1/R_1) \sqrt{L_1/C_1}$) that is typically two orders of magnitude lower than that of a quartz crystal. The lower Q makes for a faster startup of the oscillator and for a less closely controlled frequency (meaning that circuitry external to the resonator will have more influence on the frequency than with a quartz crystal).

Another major difference is that the shunt capacitance of the ceramic resonator is an order of magnitude higher than C_0 of the quartz crystal and more dependent on the frequency of the resonator.

The implications of these differences are not all obvious, but some will be indicated in the section on Oscillator Calculations.

Specifications for Ceramic Resonators

Ceramic resonators are easier to specify than quartz crystals. All the vendor wants to know is the desired

frequency and the chip you want it to work with. They'll supply the resonators, a circuit diagram showing the positions and values of other external components that may be required and a guarantee that the circuit will work properly at the specified frequency.

OSCILLATOR DESIGN CONSIDERATIONS

Designers of microcontroller systems have a number of options to choose from for clocking the system. The main decision is whether to use the "on-chip" oscillator or an external oscillator. If the choice is to use the on-chip oscillator, what kinds of external components are needed to make it operate as advertised? If the choice is to use an external oscillator, what type of oscillator should it be?

The decisions have to be based on both economic and technical requirements. In this section we'll discuss some of the factors that should be considered.

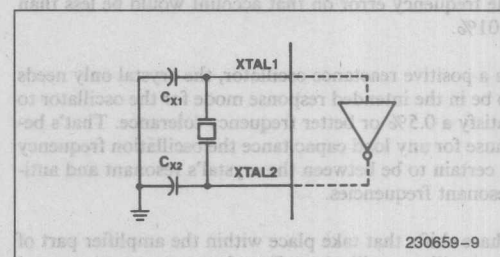


Figure 8. Using the "On-Chip" Oscillator

On-Chip Oscillators

In most cases, the on-chip amplifier with the appropriate external components provides the most economical solution to the clocking problem. Exceptions may arise in severe environments when frequency tolerances are tighter than about 0.01%.

The external components that need to be added are a positive reactance (normally a crystal or ceramic resonator) and the two capacitors C_{X1} and C_{X2} , as shown in Figure 8.

Crystal Specifications

Specifications for an appropriate crystal are not very critical, unless the frequency is. Any fundamental-mode crystal of medium or better quality can be used.

We are often asked what maximum crystal resistance should be specified. The best answer to this question is the lower the better, but use what's available. The crystal resistance will have some effect on start-up time and steady-state amplitude, but not so much that it can't be compensated for by appropriate selection of the capacitances C_{X1} and C_{X2} .

Similar questions are asked about specifications of load capacitance and shunt capacitance. The best advice we can give is to understand what these parameters mean and how they affect the operation of the circuit (that being the purpose of this Application Note), and then decide for yourself if such specifications are meaningful in your application or not. Normally, they're not, unless your frequency tolerances are tighter than about 0.1%.

Part of the problem is that crystal manufacturers are accustomed to talking "ppm" tolerances with radio engineers and simply won't take your order until you've filled out their list of specifications. It will help if you define your actual frequency tolerance requirements, both for yourself and to the crystal manufacturer. Don't pay for 0.003% crystals if your actual frequency tolerance is 1%.

Oscillation Frequency

The oscillation frequency is determined 99.5% by the crystal and up to about 0.5% by the circuit external to the crystal. The on-chip amplifier has little effect on the frequency, which is as it should be, since the amplifier parameters are temperature and process dependent.

The influence of the on-chip amplifier on the frequency is by means of its input and output (pin-to-ground) capacitances, which parallel C_{X1} and C_{X2} , and the XTAL1-to-XTAL2 (pin-to-pin) capacitance, which parallels the crystal. The input and pin-to-pin capacitances are about 7 pF each. Internal phase deviations from the nominal 180° can be modeled as an output capacitance of 25 to 30 pF. These deviations from the ideal have less effect in the positive reactance oscillator (with the inverting amplifier) than in a comparable series resonant oscillator (with the noninverting amplifier) for two reasons: first, the effect of the output capacitance is lessened, if not swamped, by the off-chip capacitor; secondly, the positive reactance oscillator is less sensitive, frequency-wise, to such phase errors.

Selection of C_{X1} and C_{X2}

Optimal values for the capacitors C_{X1} and C_{X2} depend on whether a quartz crystal or ceramic resonator

is being used, and also on application-specific requirements on start-up time and frequency tolerance.

Start-up time is sometimes more critical in microcontroller systems than frequency stability, because of various reset and initialization requirements.

Less commonly, accuracy of the oscillator frequency is also critical, for example, when the oscillator is being used as a time base. As a general rule, fast start-up and stable frequency tend to pull the oscillator design in opposite directions.

Considerations of both start-up time and frequency stability over temperature suggest that C_{X1} and C_{X2} should be about equal and at least 20 pF. (But they don't have to be either.) Increasing the value of these capacitances above some 40 or 50 pF improves frequency stability. It also tends to increase the start-up time. There is a maximum value (several hundred pF, depending on the value of R_1 of the quartz or ceramic resonator) above which the oscillator won't start up at all.

If the on-chip amplifier is a simple inverter, such as in the 8051, the user can select values for C_{X1} and C_{X2} between some 20 and 100 pF, depending on whether start-up time or frequency stability is the more critical parameter in a specific application. If the on-chip amplifier is a Schmitt Trigger, such as in the 8048, smaller values of C_{X1} must be used (5 to 30 pF), in order to prevent the oscillator from running in a relaxation mode.

Later sections in this Application Note will discuss the effects of varying C_{X1} and C_{X2} (as well as other parameters), and will have more to say on their selection.

Placement of Components

Noise glitches arriving at XTAL1 or XTAL2 pins at the wrong time can cause a miscount in the internal clock-generating circuitry. These kinds of glitches can be produced through capacitive coupling between the oscillator components and PCB traces carrying digital signals with fast rise and fall times. For this reason, the oscillator components should be mounted close to the chip and have short, direct traces to the XTAL1, XTAL2, and VSS pins.

Clocking Other Chips

There are times when it would be desirable to use the on-chip oscillator to clock other chips in the system.

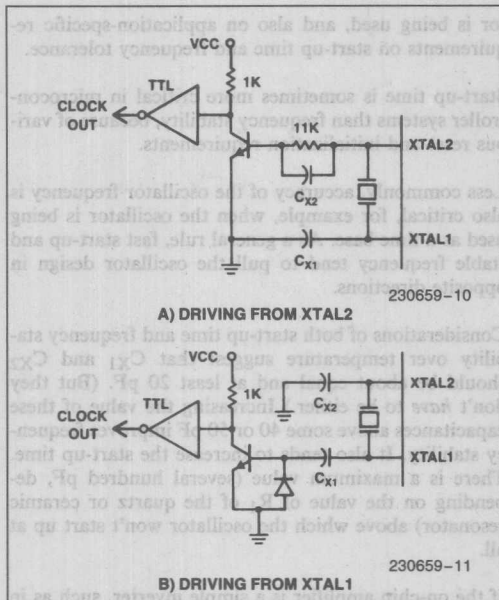


Figure 9. Using the On-Chip Oscillator to Drive Other Chips

the 8048, for example, requires that both XTAL1 and XTAL2 be driven. The 8051 can be driven that way, but the data sheet suggest the simpler method of grounding XTAL1 and driving XTAL2. For this method, the driving source must be capable of sinking some current when XTAL2 is being driven low.

Later sections in this Application Note will discuss the effects of varying C_{X1} and C_{X2} (as well as other parameters), and will have more to say on their selection.

Placement of Components

Noise glitches arriving at XTAL1 or XTAL2 pins at the wrong time can cause a miscount in the internal clock-generating circuitry. These kinds of glitches can be produced through capacitive coupling between the oscillator components and PCB traces carrying digital signals with fast rise and fall times. For this reason, the oscillator components should be mounted close to the chip and have short, direct traces to the XTAL1, XTAL2, and VSS pins.

Clocking Other Chips

There are times when it would be desirable to use the on-chip oscillator to clock other chips in the system.

This can be done if an appropriate buffer is used. A TTL buffer puts too much load on the on-chip amplifier for reliable start-up. A CMOS buffer (such as the 74HC04) can be used, if it's fast enough and if its V_{IH} and V_{IL} specs are compatible with the available signal amplitudes. Circuits such as shown in Figure 9 might also be considered for these types of applications.

Clock-related signals are available at the TO pin in the MCS-48 products, at ALE in the MCS-48 and MCS-51 lines, and the iACX-96 controllers provide a CLKOUT signal.

External Oscillators

When technical requirements dictate the use of an external oscillator, the external drive requirements for the microcontroller, as published in the data sheet, must be carefully noted. The logic levels are not in general TTL-compatible. And each controller has its idiosyncracies in this regard. The 8048, for example, requires that both XTAL1 and XTAL2 be driven. The 8051 can be driven that way, but the data sheet suggest the simpler method of grounding XTAL1 and driving XTAL2. For this method, the driving source must be capable of sinking some current when XTAL2 is being driven low.

For the external oscillator itself, there are basically two choices: ready-made and home-grown.

The oscillation frequency is determined by the crystal and up to about 0.5% by the circuit external to the crystal. The on-chip amplifier has little effect on the frequency, which is as it should be, since the amplifier parameters are temperature and process dependent.

The influence of the on-chip amplifier on the frequency is by means of its input and output (pin-to-ground) capacitances, which parallel C_{X1} and C_{X2} and the XTAL1-to-XTAL2 (pin-to-pin) capacitance, which parallels the crystal. The input and pin-to-pin capacitances are about 7 pF each. Internal phase deviations from the nominal 180° can be modeled as an output capacitance of 25 to 30 pF. These deviations from the ideal have less effect in the positive resistance oscillator (with the inverting amplifier) than in a comparable series resonant oscillator (with the noninverting amplifier) for two reasons: first, the effect of the output capacitance is lessened, if not swamped, by the off-chip capacitance; secondly, the positive resistance oscillator is less sensitive frequency-wise to such phase errors.

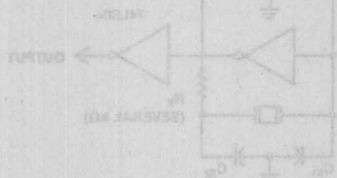
Selection of C_{X1} and C_{X2}

Optimal values for the capacitors C_{X1} and C_{X2} depend on whether a quartz crystal or ceramic resonator is being used.

TTL Crystal Clock Oscillator

The HS-100, HS-200, & HS-500 all-metal package series of oscillators are TTL compatible & fit a DIP layout. Standard electrical specifications are shown below. Variations are available for special applications.

Frequency Range: HS-100—3.5 MHz to 30 MHz
HS-200—225 KHz to 3.5 MHz
HS-500—25 MHz to 60 MHz



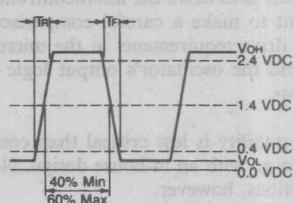
Frequency Tolerance: $\pm 0.1\%$ Overall $0^{\circ}\text{C} - 70^{\circ}\text{C}$

Hermetically Sealed Package

Mass spectrometer leak rate max.

1×10^{-8} atmos. cc/sec. of helium

Output Waveform



230659-12

INPUT				
	HS-100		HS-200	HS-500
	3.5 MHz-20 MHz	20 + MHz-30 MHz	225 KHz-4.0 MHz	25 MHz-60 MHz
Supply Voltage (V _{CC})	5V $\pm 10\%$	5V $\pm 10\%$	5V $\pm 10\%$	5V $\pm 10\%$
Supply Current (I _{CC}) max.	30 mA	40 mA	85 mA	50 mA
OUTPUT				
	HS-100		HS-200	HS-500
	3.5 MHz-20 MHz	20 + MHz-30 MHz	225 KHz-4.0 MHz	25 MHz-60 MHz
V _{OH} (Logic "1")	+2.4V min. ¹	+2.7V min. ²	+2.4V min. ¹	+2.7V min. ²
V _{OL} (Logic "0")	+0.4V max. ³	+0.5V max. ⁴	+0.4V max. ³	+0.5V max. ⁴
Symmetry	60/40% ⁵	60/40% ⁵	55/45% ⁵	60/40% ⁵
T _R , T _F (Rise & Fall Time)	< 10 ns ⁶	< 5 ns ⁶	< 15 ns ⁶	< 5 ns ⁶
Output Short				
Circuit Current	18 mA min.	40 mA min.	18 mA min.	40 mA min.
Output Load	1 to 10 TTL Loads ⁷	1 to 10 TTL Loads ⁸	1 to 10 TTL Loads ⁷	1 to 10 TTL Loads ⁸
CONDITIONS				
¹ I _O source = -400 μA max.		⁴ I _O sink = 20.00 mA max.	⁷ 1.6 mA per load	
² I _O source = -1.0 mA max.		⁵ V _O = 1.4V	⁸ 2.0 mA per load	
³ I _O sink = 16.0 mA max.		⁶ (0.4V to 2.4V)		

Figure 10. Pre-Packaged Oscillator Data*

Prepackaged oscillators are available from most crystal manufacturers, and have the advantage that the system designer can treat the oscillator as a black box whose performance is guaranteed by people who carry many years of experience in designing and building oscillators. Figure 10 shows a typical data sheet for some prepackaged oscillators. Oscillators are also available with complementary outputs.

If the oscillator is to drive the microcontroller directly, one will want to make a careful comparison between the external drive requirements in the microcontroller data sheet and the oscillator's output logic levels and test conditions.

If oscillator stability is less critical than cost, the user may prefer to go with an in-house design. Not without some precautions, however.

It's easy to design oscillators that work. Almost all of them do work, even if the designer isn't too clear on why. The key point here is that *almost* all of them work. The problems begin when the system goes into production, and marginal units commence malfunctioning in the field. Most digital designers, after all, are not very adept at designing oscillators for production.

Oscillator design is somewhat of a black art, with the quality of the finished product being very dependent on the designer's experience and intuition. For that reason the most important consideration in any design is to have an adequate preproduction test program. Preproduction tests are discussed later in this Application Note. Here we will discuss some of the design options and take a look at some commonly used configurations.

Gate Oscillators versus Discrete Devices

Digital systems designers are understandably reluctant to get involved with discrete devices and their peculiarities (biasing techniques, etc.). Besides, the component count for these circuits tends to be quite a bit higher than what a digital designer is used to seeing for that amount of functionality. Nevertheless, if there are unusual requirements on the accuracy and stability of the clock frequency, it should be noted that discrete device oscillators can be tailored to suit the exact needs of the application and perfected to a level that would be difficult for a gate oscillator to approach.

In most cases, when an external oscillator is needed, the designer tends to rely on some form of a gate oscillator. A TTL inverter with a resistor connecting the output to the input makes a suitable inverting amplifier. The resistor holds the inverter in the transition region between logical high and low, so that at least for start-up purposes the inverter is a linear amplifier.

The feedback resistance has to be quite low, however, since it must conduct current sourced by the input pin without allowing the DC input voltage to get too far above the DC output voltage. For biasing purposes, the feedback resistance should not exceed a few k-ohms. But shunting the crystal with such a low resistance does not encourage start-up.

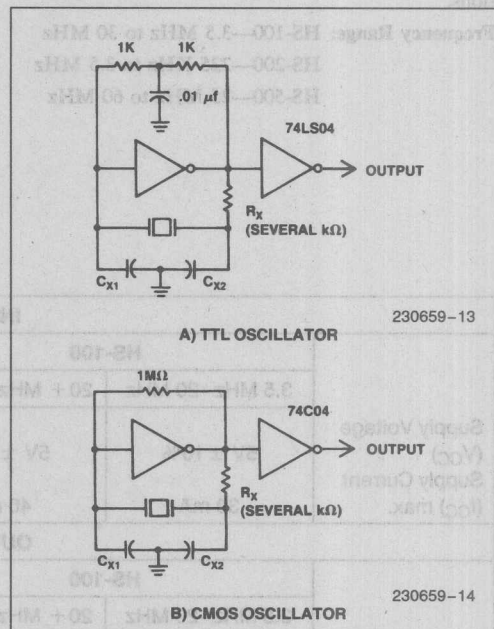


Figure 11. Commonly Used Gate Oscillators

Consequently, the configuration in Figure 11A might be suggested. By breaking R_f into two parts and AC-grounding the midpoint, one achieves the DC feedback required to hold the inverter in its active region, but without the negative signal feedback that is in effect telling the circuit *not* to oscillate. However, this biasing scheme will increase the start-up time, and relaxation-type oscillations are also possible.

A CMOS inverter, such as the 74HC04, might work better in this application, since a larger R_f can be used to hold the inverter in its linear region.

Logic gates tend to have a fairly low output resistance, which destabilizes the oscillator. For that reason a resistor R_x is often added to the feedback network, as shown in Figures 11A and B. At higher frequencies a 20 or 30 pF capacitor is sometimes used in the R_x position, to compensate for some of the internal propagation delay.

Reference 1 contains an excellent discussion of gate oscillators, and a number of design examples.

Fundamental versus Overtone Operation

It's easier to design an oscillator circuit to operate in the resonator's fundamental response mode than to design one for overtone operation. A quartz crystal whose fundamental response mode covers the desired frequency can be obtained up to some 30 MHz. For frequencies above that, the crystal might be used in an overtone mode.

Several problems arise in the design of an overtone oscillator. One is to stop the circuit from oscillating in the fundamental mode, which is what it would really rather do, for a number of reasons, involving both the amplifying device and the crystal. An additional problem with overtone operation is an increased tendency to spurious oscillations. That is because the R_1 of various spurious modes is likely to be about the same as R_1 of the intended overtone response. It may be necessary, as suggested in Reference 1, to specify a "spurious-to-main-response" resistance ratio to avoid the possibility of trouble.

Overtone oscillators are not to be taken lightly. One would be well advised to consult with an engineer who is knowledgeable in the subject during the design phase of such a circuit.

Series versus Parallel Operation

Series resonant oscillators use noninverting amplifiers. To make a noninverting amplifier out of logic gates requires that two inverters be used, as shown in Figure 12.

This type of circuit tends to be inaccurate and unstable in frequency over variations in temperature and V_{CC} . It has a tendency to oscillate at overtones, and to oscillate through C_0 of the crystal or some stray capacitance rather than as controlled by the mechanical resonance of the crystal.

The demon in series resonant oscillators is the phase shift in the amplifier. The series resonant oscillator wants more than just a "noninverting" amplifier—it wants a *zero phase-shift* amplifier. Multistage noninverting amplifiers tend to have a considerably lagging phase shift, such that the crystal reactance must be capacitive in order to bring the total phase shift around the feedback loop back up to 0. In this mode, a "12 MHz" crystal may be running at 8 or 9 MHz. One can put a capacitor in series with the crystal to relieve the crystal of having to produce all of the required phase shift, and bring the oscillation frequency closer to fs. However, to further complicate the situation, the amplifier's phase shift is strongly dependent on frequency, temperature, V_{CC} , and device sample.

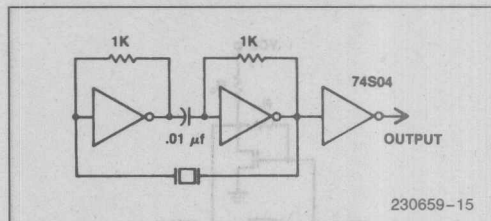


Figure 12. "Series Resonant" Gate Oscillator

Positive reactance oscillators ("parallel resonant") use inverting amplifiers. A single logic inverter can be used for the amplifier, as in Figure 11. The amplifier's phase shift is less critical, compared to a series resonant circuit, and since only one inverter is involved there's less phase error anyway. The oscillation frequency is effectively bounded by the resonant and antiresonant frequencies of the crystal itself. In addition, the feedback network includes capacitors that parallel the input and output terminals of the amplifier, thus reducing the effect of unpredictable capacitances at these points.

MORE ABOUT USING THE "ON-CHIP" OSCILLATORS

In this section we will describe the on-chip inverters on selected microcontrollers in some detail, and discuss criteria for selecting components to work with them. Future data sheets will supplement this discussion with updates and information pertinent to the use of each chip's oscillator circuitry.

Oscillator Calculations

Oscillator design, though aided by theory, is still largely an empirical exercise. The circuit is inherently nonlinear, and the normal analysis parameters vary with instantaneous voltage. In addition, when dealing with the on-chip circuitry, we have FETs being used as resistors, input protection devices, parasitic junctions, and processing variations.

Consequently, oscillator calculations are never very precise. They can be useful, however, if they will at least indicate the effects of variations in the circuit parameters on start-up time, oscillation frequency, and steady-state amplitude. Start-up time, for example, can be taken as an indication of start-up reliability. If pre-production tests indicate a possible start-up problem, a relatively inexperienced designer can at least be made aware of what parameter may be causing the marginality, and what direction to go in to fix it.

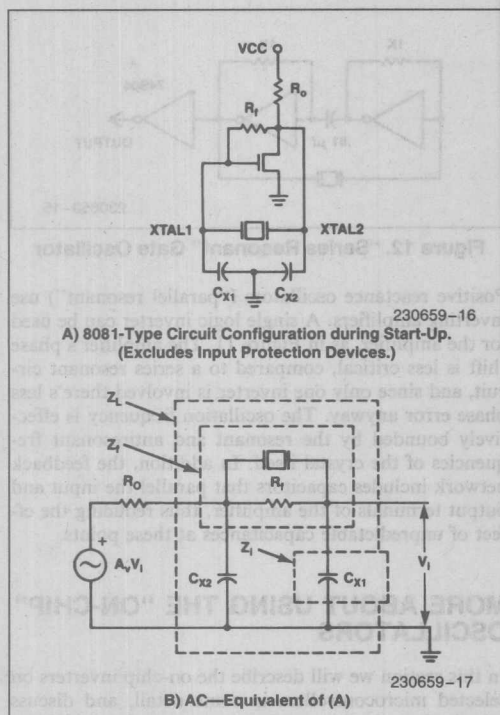


Figure 13. Oscillator Circuit Model Used in Start-Up Calculations

The analysis used here is mathematically straightforward but algebraically intractable. That means it's relatively easy to understand and program into a computer, but it will not yield a neat formula that gives, say, steady-state amplitude as a function of this or that list of parameters. A listing of a BASIC program that implements the analysis will be found in Appendix II.

When the circuit is first powered up, and before the oscillations have commenced (and if the oscillations fail to commence), the oscillator can be treated as a small signal linear amplifier with feedback. In that case, standard small-signal analysis techniques can be used to determine start-up characteristics. The circuit model used in this analysis is shown in Figure 13.

The circuit approximates that there are no high-frequency effects within the amplifier itself, such that its high-frequency behavior is dominated by the load impedance Z_L . This is a reasonable approximation for single-stage amplifiers of the type used in 8051-type devices. Then the gain of the amplifier as a function of frequency is

$$A = \frac{A_v Z_L}{Z_L + R_0}$$

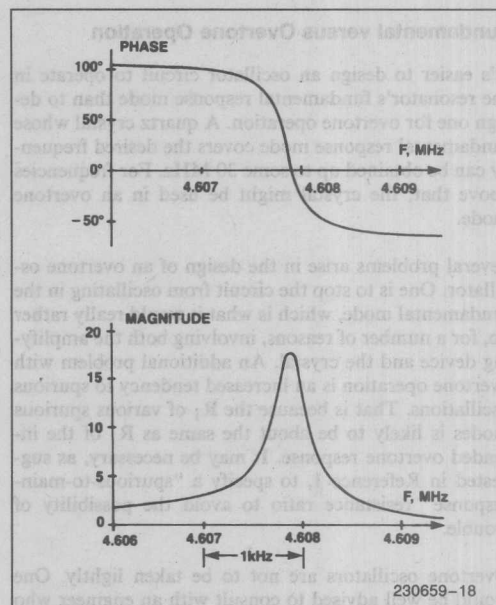


Figure 14. Loop Gain versus Frequency (4.608 MHz Crystal)

The gain of the feedback network is

$$\beta = \frac{Z_f}{Z_f + Z_L}$$

And the loop gain is

$$\beta A = \frac{Z_f}{Z_f + Z_L} \times \frac{A_v Z_L}{Z_L + R_0}$$

The impedances Z_L , Z_f , and Z_1 are defined in Figure 13B.

Figure 14 shows the way the loop gain thus calculated (using typical 8051-type parameters and a 4.608 MHz crystal) varies with frequency. The frequency of interest is the one for which the phase of the loop gain is zero. The accepted criterion for start-up is that the magnitude of the loop gain must exceed unity at this frequency. This is the frequency at which the circuit is in resonance. It corresponds very closely with the antiresonant frequency of the motional arm of the crystal in parallel with C_L .

Figure 15 shows the way the loop gain varies with frequency when the parameters of a 3.58 MHz ceramic resonator are used in place of a crystal (the amplifier parameters being typical 8051, as in Figure 14). Note the different frequency scales.

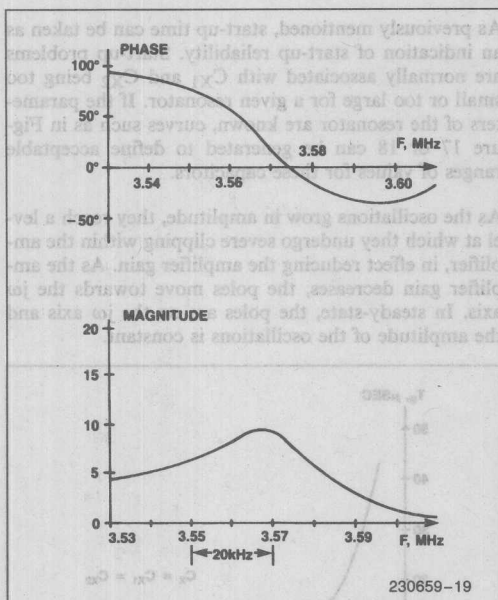


Figure 15. Loop Gain versus Frequency (3.58 MHz Ceramic)

Start-Up Characteristics

It is common, in studies of feedback systems, to examine the behavior of the closed loop gain as a function of complex frequency $s = \sigma + j\omega$; specifically, to determine the location of its poles in the complex plane. A pole is a point on the complex plane where the gain function goes to infinity. Knowledge of its location can be used to predict the response of the system to an input disturbance.

The way that the response function depends on the location of the poles is shown in Figure 16. Poles in the left-half plane cause the response function to take the form of a damped sinusoid. Poles in the right-half plane cause the response function to take the form of an exponentially growing sinusoid. In general,

$$v(t) \sim e^{at} \sin(\omega t + \theta)$$

where a is the real part of the pole frequency. Thus if the pole is in the right-half plane, a is positive and the sinusoid grows. If the pole is in the left-half plane, a is negative and the sinusoid is damped.

The same type of analysis can usefully be applied to oscillators. In this case, however, rather than trying to ensure that the poles are in the left-half plane, we would seek to ensure that they're in the *right*-half plane. An exponentially growing sinusoid is exactly what is wanted from an oscillator that has just been powered up.

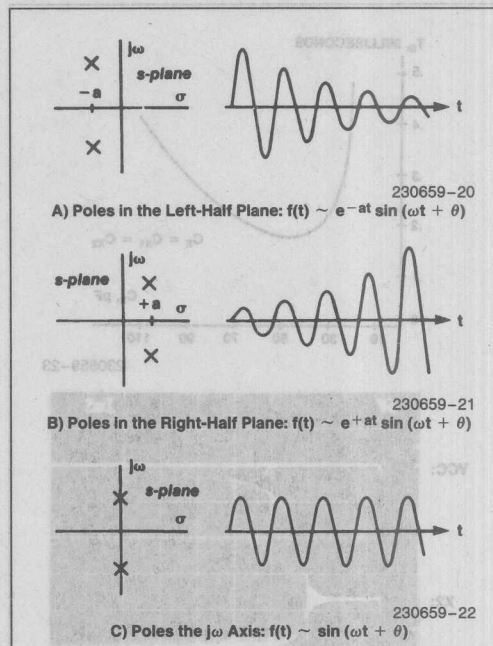


Figure 16. Do You Know Where Your Poles Are Tonight?

The gain function of interest in oscillators is $1/(1 - \beta A)$. Its poles are at the complex frequencies where $\beta A = 1 \angle 0^\circ$, because that value of βA causes the gain function to go to infinity. The oscillator will start up if the real part of the pole frequency is positive. More importantly, the *rate* at which it starts up is indicated by how *much* greater than 0 the real part of the pole frequency is.

The circuit in Figure 13B can be used to find the pole frequencies of the oscillator gain function. All that needs to be done is evaluate the impedances at complex frequencies $\sigma + j\omega$ rather than just at ω , and find the value of $\sigma + j\omega$ for which $\beta A = 1 \angle 0^\circ$. The larger that value of σ is, the faster the oscillator will start up.

Of course, other things besides pole frequencies, things like the VCC rise time, are at work in determining the start-up time. But to the extent that the pole frequencies *do* affect start-up time, we can obtain results like those in Figures 17 and 18.

To obtain these figures the pole frequencies were computed for various values of capacitance C_X from XTAL1 and XTAL2 to ground (thus $C_{X1} = C_{X2} = C_X$). Then a "time constant" for start-up was calculated as $T_s = \frac{1}{\sigma}$ where σ is the real part of the pole frequency (rad/sec), and this time constant is plotted versus C_X .

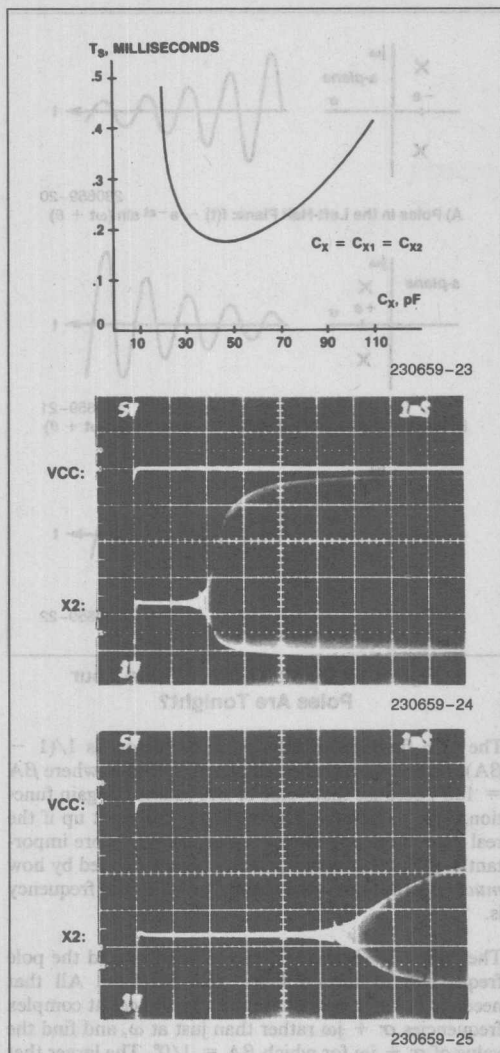


Figure 17. Oscillator Start-Up (4.608 MHz Crystal from Standard Crystal Corp.)

A short time constant means faster start-up. A long time constant means slow start-up. Observations of actual start-ups are shown in the figures. Figure 17 is for a typical 8051 with a 4.608 MHz crystal supplied by Standard Crystal Corp., and Figure 18 is for a typical 8051 with a 3.58 MHz ceramic resonator supplied by NTK Technical Ceramics, Ltd.

It can be seen in Figure 17 that, for this crystal, values of C_X between 30 and 50 pF minimize start-up time, but that the exact value in this range is not particularly important, even if the start-up time itself is critical.

As previously mentioned, start-up time can be taken as an indication of start-up reliability. Start-up problems are normally associated with C_{X1} and C_{X2} being too small or too large for a given resonator. If the parameters of the resonator are known, curves such as in Figure 17 or 18 can be generated to define acceptable ranges of values for these capacitors.

As the oscillations grow in amplitude, they reach a level at which they undergo severe clipping within the amplifier, in effect reducing the amplifier gain. As the amplifier gain decreases, the poles move towards the $j\omega$ axis. In steady-state, the poles are on the $j\omega$ axis and the amplitude of the oscillations is constant.

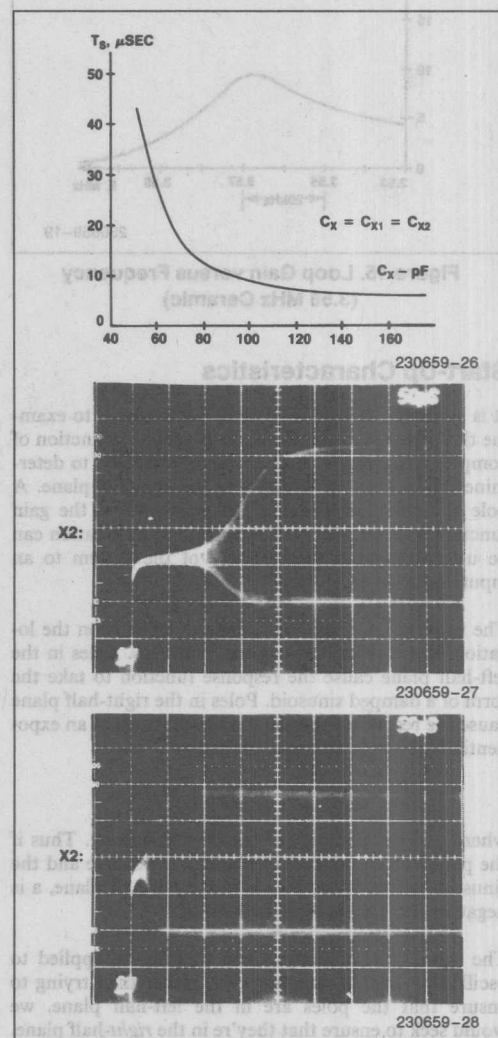


Figure 18. Oscillator Start-Up (3.58 MHz Ceramic Resonator from NTK Technical Ceramics)

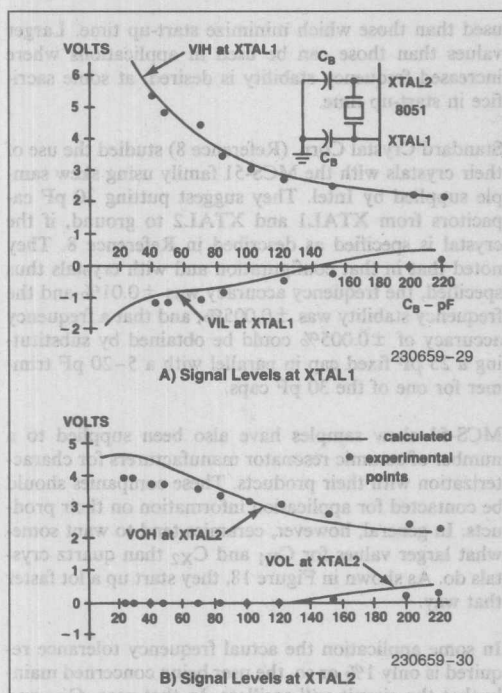


Figure 19. Calculated and Experimental Steady-State Amplitudes vs. Bulk Capacitance from XTAL1 and XTAL2 to Ground

Steady-State Characteristics

Steady-state analysis is greatly complicated by the fact that we are dealing with large signals and nonlinear circuit response. The circuit parameters vary with instantaneous voltage, and a number of clamping and clipping mechanisms come into play. Analyses that take all these things into account are too complicated to be of general use, and analyses that don't take them into account are too inaccurate to justify the effort.

There is a steady-state analysis in Appendix B that takes some of the complications into account and ignores others. Figure 19 shows the way the steady-state amplitudes thus calculated (using typical 8051 parameters and a 4.608 MHz crystal) vary with equal bulk capacitance placed from XTAL1 and XTAL2 to ground. Experimental results are shown for comparison.

The waveform at XTAL1 is a fairly clean sinusoid. Its negative peak is normally somewhat below zero, at a level which is determined mainly by the input protection circuitry at XTAL1.

The input protection circuitry consists of an ohmic resistor and an enhancement-mode FET with the gate

and source connected to ground (VSS), as shown in Figure 20 for the 8051, and in Figure 21 for the 8048. Its function is to limit the positive voltage at the gate of the input FET to the avalanche voltage of the drain junction. If the input pin is driven below VSS, the drain and source of the protection FET interchange roles, so its gate is connected to what is now the drain. In this condition the device resembles a diode with the anode connected to VSS.

There is a parasitic pn junction between the ohmic resistor and the substrate. In the ROM parts (8015, 8048, etc.) the substrate is held at approximately $-3V$ by the on-chip back-bias generator. In the EPROM parts (8751, 8748, etc.) the substrate is connected to VSS.

The effect of the input protection circuitry on the oscillator is that if the XTAL1 signal goes negative, its negative peak is clamped to $-V_{DS}$ of the protection FET in the ROM parts, and to about $-0.5V$ in the EPROM parts. These negative voltages on XTAL1 are in this application self-limiting and nondestructive.

The clamping action does, however, raise the DC level at XTAL1, which in turn tends to reduce the positive peak at XTAL2. The waveform at XTAL2 resembles a sinusoid riding on a DC level, and whose negative peaks are clipped off at zero.

Since it's normally the XTAL2 signal that drives the internal clocking circuitry, the question naturally arises as to how large this signal must be to reliably do its job. In fact, the XTAL2 signal doesn't have to meet the same V_{IH} and V_{IL} specifications that an external driver would have to. That's because as long as the oscillator is working, the on-chip amplifier is driving itself through its own 0-to-1 transition region, which is very nearly the same as the 0-to-1 transition region in the internal buffer that follows the oscillator. If some processing variations move the transition level higher or lower, the on-chip amplifier tends to compensate for it by the fact that its own transition level is correspondingly higher or lower. (In the 8096, it's the XTAL1 signal that drives the internal clocking circuitry, but the same concept applies.)

The main concern about the XTAL2 signal amplitude is an indication of the general health of the oscillator. An amplitude of less than about 2.5V peak-to-peak indicates that start-up problems could develop in some units (with low gain) with some crystals (with high R_1). The remedy is to either adjust the values of C_{X1} and/or C_{X2} or use a crystal with a lower R_1 .

The amplitudes at XTAL1 and XTAL2 can be adjusted by changing the ratio of the capacitors from XTAL1 and XTAL2 to ground. Increasing the XTAL2 capacitance, for example, decreases the amplitude at XTAL2 and increases the amplitude at XTAL1 by about the same amount. Decreasing both caps increases both amplitudes.

Pin Capacitance

Internal pin-to-ground and pin-to-pin capacitances at XTAL1 and XTAL2 will have some effect on the oscillator. These capacitances are normally taken to be in the range of 5 to 10 pF, but they are extremely difficult to evaluate. Any measurement of one such capacitance will necessarily include effects from the others. One advantage of the positive reactance oscillator is that the pin-to-ground capacitances are paralleled by external bulk capacitors, so a precise determination of their value is unnecessary. We would suggest that there is little justification for more precision than to assign them a value of 7 pF (XTAL1-to-ground and XTAL1-to-XTAL2). This value is probably not in error by more than 3 or 4 pF.

The XTAL2-to-ground capacitance is not entirely "pin capacitance," but more like an "equivalent output capacitance" of some 25 to 30 pF, having to include the effect of internal phase delays. This value will vary to some extent with temperature, processing, and frequency.

MCS®-51 Oscillator

The on-chip amplifier on the HMOS MCS-51 family is shown in Figure 20. The drain load and feedback "resistors" are seen to be field-effect transistors. The drain load FET, R_D , is typically equivalent to about 1K to 3 K-ohms. As an amplifier, the low frequency voltage gain is normally between -10 and -20, and the output resistance is effectively R_D .

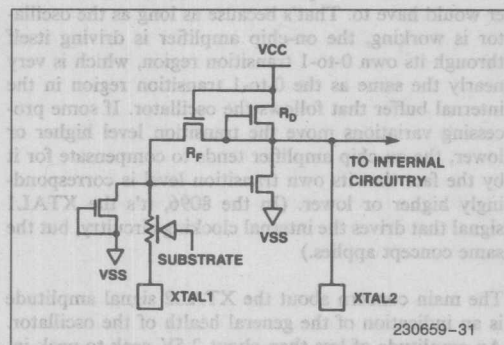


Figure 20. MCS®-51 Oscillator Amplifier

The 80151 oscillator is normally used with equal bulk capacitors placed externally from XTAL1 to ground and from XTAL2 to ground. To determine a reasonable value of capacitance to use in these positions, given a crystal of ceramic resonator of known parameters, one can use the BASIC analysis in Appendix II to generate curves such as in Figures 17 and 18. This procedure will define a range of values that will minimize start-up time. We don't suggest that smaller values be

used than those which minimize start-up time. Larger values than those can be used in applications where increased frequency stability is desired, at some sacrifice in start-up time.

Standard Crystal Corp. (Reference 8) studied the use of their crystals with the MCS-51 family using skew sample supplied by Intel. They suggest putting 30 pF capacitors from XTAL1 and XTAL2 to ground, if the crystal is specified as described in Reference 8. They noted that in that configuration and with crystals thus specified, the frequency accuracy was $\pm 0.01\%$ and the frequency stability was $\pm 0.005\%$, and that a frequency accuracy of $\pm 0.005\%$ could be obtained by substituting a 25 pF fixed cap in parallel with a 5-20 pF trimmer for one of the 30 pF caps.

MCS-51 skew samples have also been supplied to a number of ceramic resonator manufacturers for characterization with their products. These companies should be contacted for application information on their products. In general, however, ceramics tend to want somewhat larger values for C_{X1} and C_{X2} than quartz crystals do. As shown in Figure 18, they start up a lot faster that way.

In some application the actual frequency tolerance required is only 1% or so, the user being concerned mainly that the circuit will oscillate. In that case, C_{X1} and C_{X2} can be selected rather freely in the range of 20 to 80 pF.

As you can see, "best" values for these components and their tolerances are strongly dependent on the application and its requirements. In any case, their suitability should be verified by environmental testing before the design is submitted to production.

MCS®-48 Oscillator

The NMOS and HMOS MCS-48 oscillator is shown in Figure 21. It differs from the 8051 in that its inverting

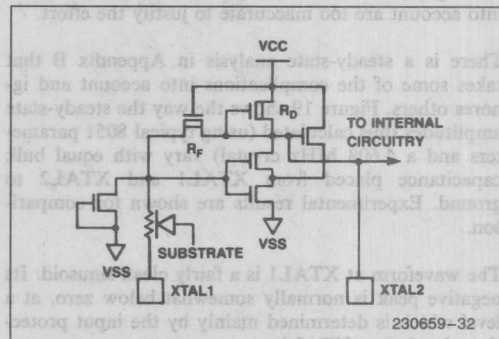


Figure 21. MCS®-48 Oscillator Amplifier

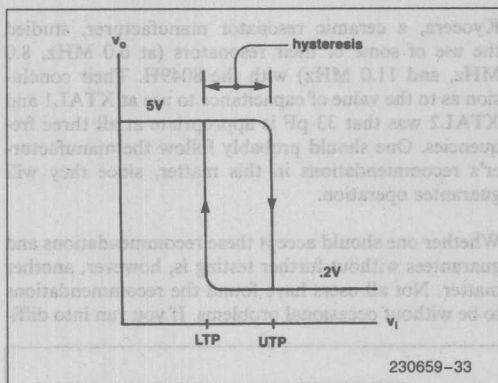


Figure 22. Schmitt Trigger Characteristic

amplifier is a Schmitt Trigger. This configuration was chosen to prevent crosstalk from the TO pin, which is adjacent to the XTAL1 pin.

All Schmitt Trigger circuits exhibit a hysteresis effect, as shown in Figure 22. The hysteresis is what makes it less sensitive to noise. The same hysteresis allows any Schmitt Trigger to be used as a relaxation oscillator. All you have to do is connect a resistor from output to input, and a capacitor from input to ground, and the circuit oscillates in a relaxation mode as follows.

If the Schmitt Trigger output is at a logic high, the capacitor commences charging through the feedback resistor. When the capacitor voltage reaches the upper trigger point (UTP), the Schmitt Trigger output switches to a logic low and the capacitor commences discharging through the same resistor. When the capacitor voltage reaches the lower trigger point (LTP), the Schmitt Trigger output switches to a logic high again, and the sequence repeats. The oscillation frequency is determined by the RC time constant and the hysteresis voltage, UTP-LTP.

The 8048 can oscillate in this mode. It has an internal feedback resistor. All that's needed is an external capacitor from XTAL1 to ground. In fact, if a smaller external feedback resistor is added, an 8048 system could be designed to run in this mode. *Do it at your own risk!* This mode of operation is not tested, specified, documented, or encouraged in any way by Intel for the 8048. Future steppings of the device might have a different type of inverting amplifier (one more like the 8051). The CHMOS members of the MCS-48 family do not use a Schmitt Trigger as the inverting amplifier.

Relaxation oscillations in the 8048 must be avoided, and this is the major objective in selecting the off-chip components needed to complete the oscillator circuit.

When an 8048 is powered up, if VCC has a short rise time, the relaxation mode starts first. The frequency is normally about 50 KHz. The resonator mode builds

more slowly, but it eventually takes over and dominates the operation of the circuit. This is shown in Figure 23A.

Due to processing variations, some units seem to have a harder time coming out of the relaxation mode, particularly at low temperatures. In some cases the resonator oscillations may fail entirely, and leave the device in the relaxation mode. Most units will stick in the relaxation mode at any temperature if C_{X1} is larger than about 50 pF. Therefore, C_{X1} should be chosen with some care, particularly if the system must operate at lower temperatures.

One method that has proven effective in all units to -40°C is to put 5 pF from XTAL1 to ground and 20 pF from XTAL2 to ground. Unfortunately, while this method does discourage the relaxation mode, it is not an optimal choice for the resonator mode. For one thing, it does not swamp the pin capacitance. Also, it makes for a rather high signal level at XTAL1 (8 or 9 volts peak-to-peak).

The question arises as to whether that level of signal at XTAL1 might damage the chip. Not to worry. The negative peaks are self-limiting and nondestructive. The positive peaks could conceivably damage the oxide, but in fact, NMOS chips (eg, 8048) and HMOS chips (eg, 8048H) are tested to a much higher voltage than that. The technology trend, of course, is to thinner oxides, as the devices shrink in size. For an extra margin of safety, the HMOS II chips (eg, 8048AH) have an internal diode clamp at XTAL1 to VCC.

In reality, C_{X1} doesn't have to be quite so small to avoid relaxation oscillations, if the minimum operating temperature is not -40°C . For less severe temperature requirements, values of capacitance selected in much the same way as for an 8051 can be used. The circuit should be tested, however, at the system's lowest temperature limit.

Additional security against relaxation oscillations can be obtained by putting a 1M-ohm (or larger) resistor from XTAL1 to VCC. Pulling up the XTAL1 pin this way seems to discourage relaxation oscillations as effectively as any other method (Figure 23B).

Another thing that discourages relaxation oscillations is low VCC. The resonator mode, on the other hand is much less sensitive to VCC. Thus if VCC comes up relatively slowly (several milliseconds rise time), the resonator mode is normally up and running before the relaxation mode starts (in fact, before VCC has even reached operating specs). This is shown in Figure 23C.

A secondary effect of the hysteresis is a shift in the oscillation frequency. At low frequencies, the output signal from an inverter without hysteresis leads (or lags) the input by 180 degrees. The hysteresis in a Schmitt Trigger, however, causes the output to lead the

degrees), by an amount that depends on the signal amplitude, as shown in Figure 24. At higher frequencies, there are additional phase shifts due to the various reactances in the circuit, but the phase shift due to the hysteresis is still present. Since the total phase shift in the oscillator's loop gain is necessarily 0 or 360 degrees, it is apparent that as the oscillations build up, the frequency has to change to allow the reactances to compensate for the hysteresis. In normal operation, this additional phase shift due to hysteresis does not exceed a few degrees, and the resulting frequency shift is negligible.

the use of some of their resonators (at 6.0 MHz, 8.0 MHz, and 11.0 MHz) with the 8049H. Their conclusion as to the value of capacitance to use at XTAL1 and XTAL2 was that 33 pF is appropriate at all three frequencies. One should probably follow the manufacturer's recommendations in this matter, since they will guarantee operation.

Whether one should accept these recommendations and guarantees without further testing is, however, another matter. Not all users have found the recommendations to be without occasional problems. If you run into diffi-

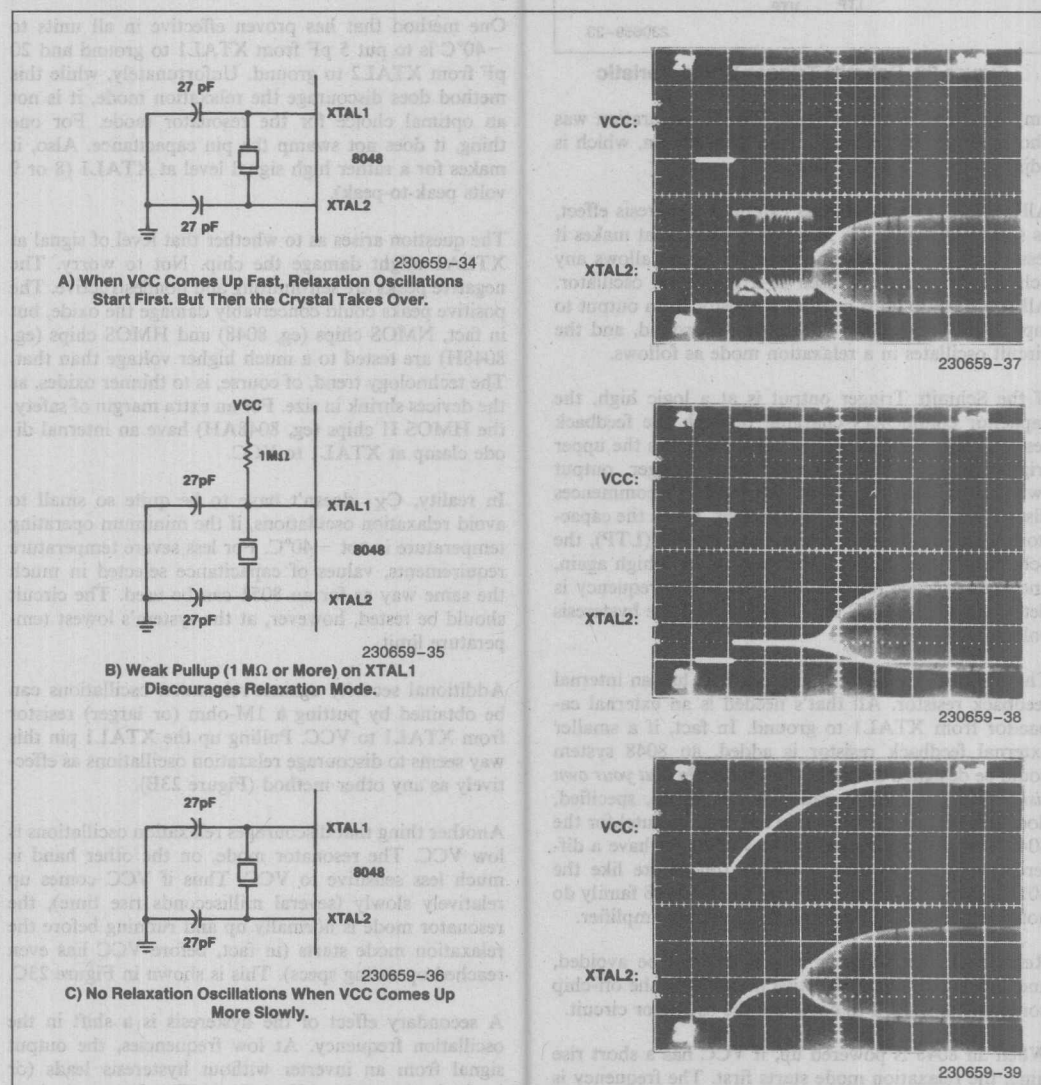


Figure 23. Relaxation Oscillations in the 8048

culties using their recommendations, both Intel and the ceramic resonator manufacturer want to know about it. It is to their interest, and ours, that such problems be resolved.

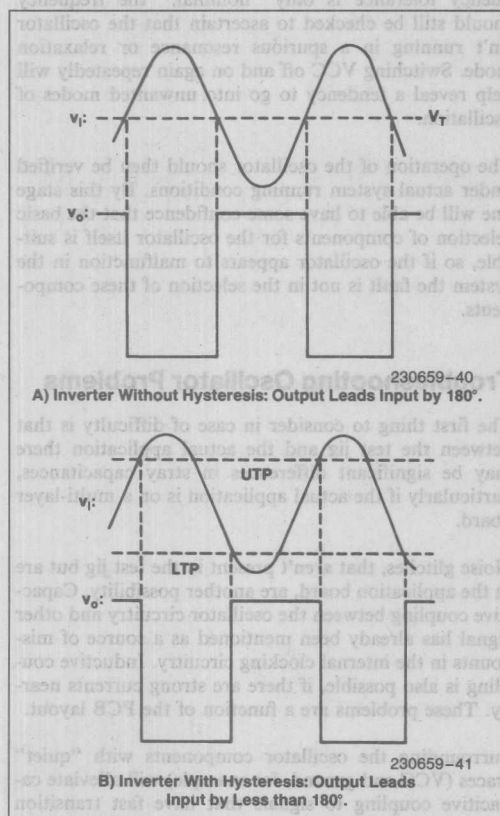


Figure 24. Amplitude-Dependent Phase Shift in Schmitt Trigger

Preproduction Tests

An oscillator design should never be considered ready for production until it has proven its ability to function acceptably well under worst-case environmental conditions and with parameters at their worst-case tolerance limits. Unexpected temperature effects in parts that may already be near their tolerance limits can prevent start-up of an oscillator that works perfectly well on the bench. For example, designers often overlook temperature effects in ceramic capacitors. (Some ceramics are down to 50% of their room-temperature values at -20°C and $+60^{\circ}\text{C}$). The problem here isn't just one of frequency stability, but also involves start-up time and steady-state amplitude. There may also be temperature effects in the resonator and amplifier.

It will be helpful to build a test jig that will allow the oscillator circuit to be tested independently of the rest of the system. Both start-up and steady-state characteristics should be tested. Figure 25 shows the circuit that

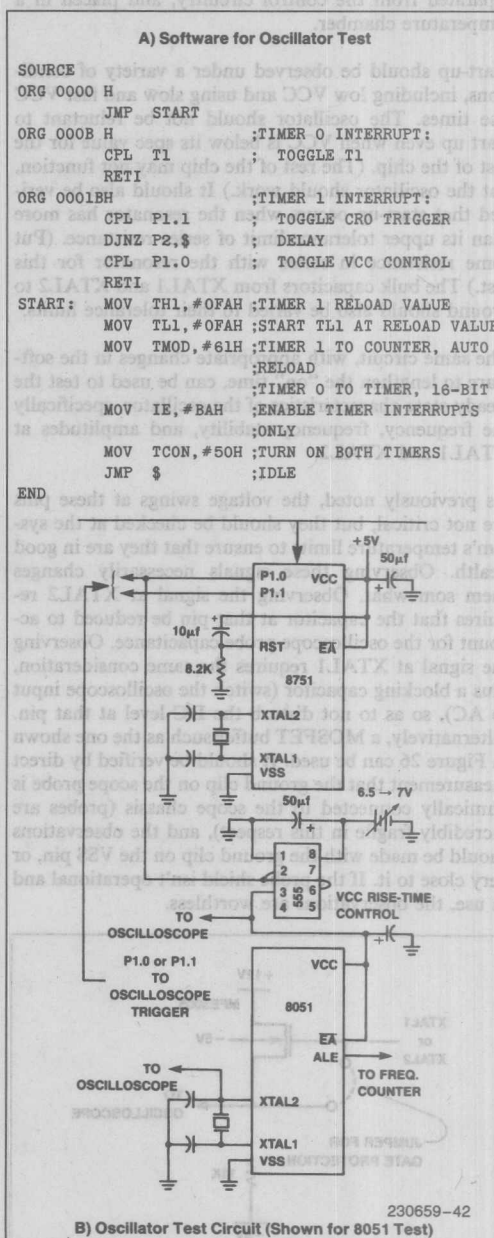


Figure 25. Oscillator Test Circuit and Software

was used to obtain the oscillator start-up photographs in this Application Note. This circuit or a modified version of it would make a convenient test vehicle. The oscillator and its relevant components can be physically separated from the control circuitry, and placed in a temperature chamber.

Start-up should be observed under a variety of conditions, including low VCC and using slow and fast VCC rise times. The oscillator should not be reluctant to start up even when VCC is below its spec value for the rest of the chip. (The rest of the chip may not function, but the oscillator should work.) It should also be verified that start-up occurs when the resonator has more than its upper tolerance limit of series resistance. (Put some resistance in series with the resonator for this test.) The bulk capacitors from XTAL1 and XTAL2 to ground should also be varied to their tolerance limits.

The same circuit, with appropriate changes in the software to lengthen the "on" time, can be used to test the steady-state characteristics of the oscillator, specifically the frequency, frequency stability, and amplitudes at XTAL1 and XTAL2.

As previously noted, the voltage swings at these pins are not critical, but they should be checked at the system's temperature limits to ensure that they are in good health. Observing these signals necessarily changes them somewhat. Observing the signal at XTAL2 requires that the capacitor at that pin be reduced to account for the oscilloscope probe capacitance. Observing the signal at XTAL1 requires the same consideration, plus a blocking capacitor (switch the oscilloscope input to AC), so as to not disturb the DC level at that pin. Alternatively, a MOSFET buffer such as the one shown in Figure 26 can be used. It should be verified by direct measurement that the ground clip on the scope probe is ohmically connected to the scope chassis (probes are incredibly fragile in this respect), and the observations should be made with the ground clip on the VSS pin, or very close to it. If the probe shield isn't operational and in use, the observations are worthless.

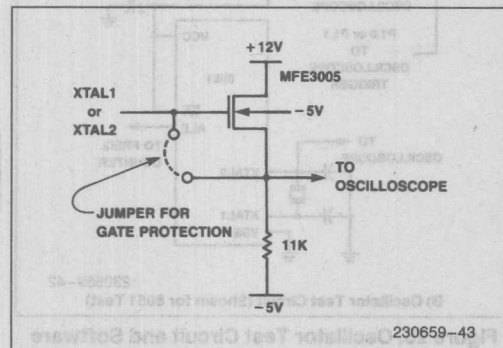


Figure 26. MOSFET Buffer for Observing Oscillator Signals

Frequency checks should be made with only the oscillator circuitry connected to XTAL1 and XTAL2. The ALE frequency can be counted, and the oscillator frequency derived from that. In systems where the frequency tolerance is only "nominal," the frequency should still be checked to ascertain that the oscillator isn't running in a spurious resonance or relaxation mode. Switching VCC off and on again repeatedly will help reveal a tendency to go into unwanted modes of oscillation.

The operation of the oscillator should then be verified under actual system running conditions. By this stage one will be able to have some confidence that the basic selection of components for the oscillator itself is suitable, so if the oscillator appears to malfunction in the system the fault is not in the selection of these components.

Troubleshooting Oscillator Problems

The first thing to consider in case of difficulty is that between the test jig and the actual application there may be significant differences in stray capacitances, particularly if the actual application is on a multi-layer board.

Noise glitches, that aren't present in the test jig but are in the application board, are another possibility. Capacitive coupling between the oscillator circuitry and other signal has already been mentioned as a source of miscounts in the internal clocking circuitry. Inductive coupling is also possible, if there are strong currents nearby. These problems are a function of the PCB layout.

Surrounding the oscillator components with "quiet" traces (VCC and ground, for example) will alleviate capacitive coupling to signals that have fast transition times. To minimize inductive coupling, the PCB layout should minimize the areas of the loops formed by the oscillator components. These are the loops that should be checked:

- XTAL1 through the resonator to XTAL2;
- XTAL1 through C_{X1} to the VSS pin;
- XTAL2 through C_{X2} to the VSS pin.

It is not unusual to find that the grounded ends of C_{X1} and C_{X2} eventually connect up to the VSS pin only after looping around the farthest ends of the board. Not good.

Finally, it should not be overlooked that software problems sometimes imitate the symptoms of a slow-starting oscillator or incorrect frequency. Never underestimate the perversity of a software problem.

REFERENCES

1. Frerking, M. E., *Crystal Oscillator Design and Temperature Compensation*, Van Nostrand Reinhold, 1978.
2. Bottom, V., "The Crystal Unit as a Circuit Component," Ch. 7, *Introduction to Quartz Crystal Unit Design*, Van Nostrand Reinhold, 1982.
3. Parzen, B., *Design of Crystal and Other Harmonic Oscillators*, John Wiley & Sons, 1983.
4. Holmbeck, J. D., "Frequency Tolerance Limitations with Logic Gate Clock Oscillators, 31st Annual Frequency Control Symposium, June, 1977.
5. Roberge, J. K., "Nonlinear Systems," Ch. 6, *Operational Amplifiers: Theory and Practice*, Wiley, 1975.
6. Eaton, S. S. *Timekeeping Advances Through COS/MOS Technology*, RCA Application Note ICAN-6086.
7. Eaton, S. S., *Micropower Crystal-Controlled Oscillator Design Using RCA COS/MOS Inverters*, RCA Application Note ICAN-6539.
8. Fisher, J. B., *Crystal Specifications for the Intel 8031/8051/8751 Microcontrollers*, Standard Crystal Corp. Design Data Note #2F.
9. Murata Mfg. Co., Ltd., *Ceramic Resonator "Ceralock" Application Manual*.
10. Kyoto Ceramic Co., Ltd., *Adaptability Test Between Intel 8049H and Kyocera Ceramic Resonators*.
11. Kyoto Ceramic Co., Ltd., *Technical Data on Ceramic Resonator Model KBR-6.0M, KBR-8.0M, KBR-11.0M Application for 8051 (Intel)*.
12. NTK Technical Ceramic Division, NGK Spark Plug Co., Ltd., *NTKK Ceramic Resonator Manual*.

APPENDIX A QUARTZ AND CERAMIC RESONATOR FORMULAS

Based on the equivalent circuit of the crystal, the impedance of the crystal is

$$Z_{XTAL} = \frac{(R_1 + j\omega L_1 + 1/j\omega C_1)(1/j\omega C_0)}{R_1 + j\omega L_1 + 1/j\omega C_1 + 1/j\omega C_0}$$

After some algebraic manipulation, this calculation can be written in the form

$$Z_{XTAL} = \frac{1}{j\omega(C_1 + C_0)} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T}$$

where C_T is the capacitance of C_1 in series with C_0 :

$$C_T = \frac{C_1 C_0}{C_1 + C_0}$$

The impedance of the crystal in parallel with an external load capacitance C_L is the same expression, but with $C_0 + C_L$ substituted for C_0 :

$$Z_{XTAL} \parallel C_L = \frac{1}{j\omega(C_1 + C_0 + C_L)} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C'_T + j\omega R_1 C'_T}$$

where C'_T is the capacitance of C_1 in series with $(C_0 + C_L)$:

$$C'_T = \frac{C_1(C_0 + C_L)}{C_1 + C_0 + C_L}$$

The impedance of the crystal in series with the load capacitance is

$$\begin{aligned} Z_{XTAL} + C_L &= Z_{XTAL} + \frac{1}{j\omega C_L} \\ &= \frac{C_L + C_1 + C_0}{j\omega C_L(C_1 + C_0)} \cdot \frac{1 - \omega^2 L_1 C'_T + j\omega R_1 C'_T}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T} \end{aligned}$$

where C_T and C'_T are as defined above.

The phase angles of these impedances are readily obtained from the impedance expressions themselves:

$$\begin{aligned} \theta_{XTAL} &= \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1} \\ &\quad - \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} - \frac{\pi}{2} \end{aligned}$$

$$\theta_{XTAL} \parallel C_L = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1}$$

$$- \arctan \frac{\omega R_1 C'_T}{1 - \omega^2 L_1 C'_T} - \frac{\pi}{2}$$

$$\theta_{XTAL} + C_L = \arctan \frac{\omega R_1 C'_T}{1 - \omega^2 L_1 C'_T}$$

$$- \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} - \frac{\pi}{2}$$

The resonant ("series resonant") frequency is the frequency at which the phase angle is zero and the impedance is low. The antiresonant ("parallel resonant") frequency is the frequency at which the phase angle is zero and the impedance is high.

Each of the above θ -expressions contains two arctan functions. Setting the denominator of the argument of the first arctan function to zero gives (approximately) the "series resonant" frequency for that configuration. Setting the denominator of the argument of the second arctan function to zero gives (approximately) the "parallel resonant" frequency for that configuration.

For example, the resonant frequency of the crystal is the frequency at which

$$1 - \omega^2 L_1 C_1 = 0$$

Thus

$$\omega_s = \frac{1}{\sqrt{L_1 C_1}}$$

or

$$f_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

It will be noted that the series resonant frequency of the "XTAL + CL" configuration (crystal in series with CL) is the same as the parallel resonant frequency of the "XTAL||CL" configuration (crystal in parallel with CL). This is the frequency at which

$$1 - \omega^2 L_1 C'_T = 0$$

Thus

$$\omega_a = \frac{1}{\sqrt{L_1 C'_T}}$$

or

$$f_a = \frac{1}{2\pi\sqrt{L_1 C'_T}}$$

This fact is used by crystal manufacturers in the process of calibrating a crystal to a specified load capacitance.

By subtracting the resonant frequency of the crystal from its antiresonant frequency, one can calculate the range of frequencies over which the crystal reactance is positive:

$$f_a - f_s = f_s \left(\sqrt{1 + \frac{C_1}{C_0}} - 1 \right)$$

Given typical values for C_1 and C_0 , this range can hardly exceed 0.5% of f_s . Unless the inverting amplifier in the positive reactance oscillator is doing something very strange indeed, the oscillation frequency is bound to be accurate to that percentage whether the crystal was calibrated for series operation or to any unspecified load capacitance.

Equivalent Series Resistance

ESR is the real part of Z_{XTAL} at the oscillation frequency. The oscillation frequency is the parallel resonant frequency of the "XTAL||CL" configuration (which is the same as the series resonant frequency of the "XTAL + CL" configuration). Substituting this frequency into the Z_{XTAL} expression yields, after some algebraic manipulation,

$$ESR = \frac{R_1 \left(\frac{C_0 + C_L}{C_L} \right)^2}{1 + \omega^2 C_L^2 \left(\frac{C_0 + C_L}{C_L} \right)^2}$$

$$\approx R_1 \left(1 + \frac{C_0}{C_L} \right)^2$$

Drive Level

The power dissipated by the crystal is $I_1^2 R_1$, where I_1 is the RMS current in the motional arm of the crystal. This current is given by $V_x / |Z_1|$, where V_x is the RMS voltage across the crystal, and $|Z_1|$ is the magnitude of the impedance of the motional arm. At the oscillation frequency, the motional arm is a positive (inductive) reactance in parallel resonance with $(C_0 + C_L)$. Therefore $|Z_1|$ is approximately equal to the magnitude of the reactance of $(C_0 + C_L)$:

$$|Z_1| = \frac{1}{2\pi f(C_0 + C_L)}$$

where f is the oscillation frequency. Then,

$$P = I_1^2 R_1 = \left(\frac{V_x}{|Z_1|} \right)^2 R_1$$

$$= [2\pi f (C_0 + C_L) V_x]^2 R_1$$

The waveform of the voltage across the crystal (XTAL1 to XTAL2) is approximately sinusoidal. If its peak value is V_{CC} , then V_x is $V_{CC}/\sqrt{2}$. Therefore,

$$P = 2R_1 [\pi f (C_0 + C_L) V_{CC}]^2$$

APPENDIX B OSCILLATOR ANALYSIS PROGRAM

The program is written in BASIC. BASIC is excruciatingly slow, but it has some advantages. For one thing, more people know BASIC than FORTRAN. In addition, a BASIC program is easy to develop, modify, and "fiddle around" with. Another important advantage is that a BASIC program can run on practically any small computer system.

Its slowness is a problem, however. For example, the routine which calculates the "start-up time constant" discussed in the text may take several hours to complete. A person who finds this program useful may prefer to convert it to FORTRAN, if the facilities are available.

Limitations of the Program

The program was developed with specific reference to 8051-type oscillator circuitry. That means the on-chip amplifier is a simple inverter, and not a Schmitt Trigger. The 8096, the 80C51, the 80C48 and 80C49 all have simple inverters. The 8096 oscillator is almost identical to the 8051, differing mainly in the input protection circuitry. The CHMOS amplifiers have somewhat different parameters (higher gain, for example), and different transition levels than the 8051.

The MCS-48 family is specifically included in the program only to the extent that the input-output curve used in the steady-state analysis is that of a Schmitt Trigger, if the user identifies the device under analysis as an MCS-48 device. The analysis does not include the voltage dependent phase shift of the Schmitt Trigger.

The clamping action of the input protection circuitry is important in determining the steady-state amplitudes. The steady-state routine accounts for it by setting the negative peak of the XTAL1 signal at a level which depends on the amplitude of the XTAL1 signal in accordance with experimental observations. It's an exercise in curve-fitting. A user may find a different type of curve works better. Later steppings of the chips may behave differently in this respect, having somewhat different types of input protection circuitry.

It should be noted that the analysis ignores a number of important items, such as high-frequency effects in the on-chip circuitry. These effects are difficult to predict, and are no doubt dependent on temperature, frequency, and device sample. However, they can be simulated to a reasonable degree by adding an "output capacitance" of about 20 pF to the circuit model (i.e., in parallel with CX2) as described below.

Notes on Using the Program

The program asks the user to input values for various circuit parameters. First the crystal (or ceramic resonator) parameters are asked for. These are R1, L1, C1, and C0. The manufacturer can supply these values for selected samples. To obtain any kind of correlation between calculation and experiment, the values of these parameters must be known for the specific sample in the test circuit. The value that should be entered for C0 is the C0 of the crystal itself plus an estimated 7 pF to account for the XTAL1-to-XTAL2 pin capacitance, plus any other stray capacitance paralleling the crystal that the user may feel is significant enough to be included.

Then the program asks for the values of the XTAL1-to-ground and XTAL2-to-ground capacitances. For CXTAL1, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance. For CXTAL2, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance plus about 20 pF to simulate high-frequency roll-off and phase shifts in the on-chip circuitry.

Next the program asks for values for the small-signal parameters of the on-chip amplifier. Typically, for the 8051/8751,

Amplifier Gain Magnitude	= 15
Feedback Resistance	= 2300 K Ω
Output Resistance	= 2 K Ω

The same values can be used for MCS-48 (NMOS and HMOS) devices, but they are difficult to verify, because the Schmitt Trigger does not lend itself to small-signal measurements.

```

100 DEFDBL C,D,F,G,L,P,R,S,X
200 REM
300 REM *****
400 REM
500 REM
600 REM
700 REM
800 REM FNZM(R,X) = MAGNITUDE OF A COMPLEX NUMBER, |R+jX|
900 DEF FNZM(R,X) = SQR(R^2+X^2)
1000 REM
1100 REM FNZP(R,X) = ANGLE OF A COMPLEX NUMBER
1200 REM = 180/PI*ARCTAN(X/R) IF R>0
1300 REM = 180/PI*ARCTAN(X/R) + 180 IF R<0 AND X>0
1400 REM = 180/PI*ARCTAN(X/R) - 180 IF R<0 AND X<0
1500 DEF FNZP(R,X) = 180/PI*ATN(X/R) - (SGN(R)-1)*SGN(X)*90
1600 REM
1700 REM INDUCTIVE IMPEDANCE AT COMPLEX FREQUENCY S+jF (HZ)
1800 REM Z = 2*PI*S*L + j2*PI*F*L
1900 REM = FNRL(S,L) + jFNXL(F,L)
2000 DEF FNRL(S,L) = 2*PI*S*L
2100 DEF FNXL(F,L) = 2*PI*F*L
2200 REM
2300 REM CAPACITIVE IMPEDANCE AT COMPLEX FREQUENCY S+jF (HZ)
2400 REM Z = 1/[2*PI*(S+jF)*C]
2500 REM = S/[2*PI*(S^2+F^2)*C] + j(-F)/[2*PI*(S^2+F^2)*C]
2600 REM = FNRC(S,F,C) + jFNXC(S,F,C)
2700 DEF FNRC(S,F,C) = S/[2*PI*(S^2+F^2)*C]
2800 DEF FNXC(S,F,C) = -F/[2*PI*(S^2+F^2)*C]
2900 REM
3000 REM RATIO OF TWO COMPLEX NUMBERS
3100 REM RA+jXA RA*RB+XA*XB XA*RB-RA*XB
3200 REM ----- = ----- + j -----
3300 REM RB+jXB RB^2+XB^2 RB^2+XB^2
3400 REM = FNRR(RA,XA,RB,XB) + jFNXR(RA,XA,RB,XB)
3500 DEF FNRR(RA,XA,RB,XB) = (RA*RB+XA*XB)/(RB^2+XB^2)
3600 DEF FNXR(RA,XA,RB,XB) = (XA*RB-RA*XB)/(RB^2+XB^2)
3700 REM
3800 REM PRODUCT OF TWO COMPLEX NUMBERS
3900 REM (RA+jXA)*(RB+jXB) = RA*RB-XA*XB + j(XA*RB+RA*XB)
4000 REM = FNRM(RA,XA,RB,XB) + jFNXM(RA,XA,RB,XB)
4100 DEF FNRM(RA,XA,RB,XB) = RA*RB - XA*XB
4200 DEF FNXM(RA,XA,RB,XB) = XA*RB + RA*XB
4300 REM
4400 REM
4500 REM PARALLEL IMPEDANCES
4600 REM (RA+jXA) || (RB+jXB) = (RA+jXA)*(RB+jXB) / (RA+RB + j(XA+XB))
4700 REM = (RA*RB+XA*XB) / (RA+RB + j(XA+XB))
4800 REM
4900 REM
5000 REM RA*(RB^2+XB^2)+RB*(RA^2+XA^2) XA*(RB^2+XB^2)+XB*(RA^2+XA^2)
5100 REM = ----- + j -----
5200 REM (RA+RB)^2 + (XA+XB)^2 (RA+RB)^2 + (XA+XB)^2
5300 REM
5400 REM = FNRP(RA,XA,RB,XB) + jFNXP(RA,XA,RB,XB)
5500 DEF FNRP(RA,XA,RB,XB) = (RA*(RB^2+XB^2) + RB*(RA^2+XA^2)) / ((RA+RB)^2 + (XA+XB)^2)
5600 DEF FNXP(RA,XA,RB,XB) = (XA*(RB^2+XB^2) + XB*(RA^2+XA^2)) / ((RA+RB)^2 + (XA+XB)^2)
5700 REM
5800 REM *****
5900 REM
6000 REM BEGIN COMPUTATIONS
6100 REM
6200 LET PI = 3.141592654#
6300 REM
6400 REM DEFINE CIRCUIT PARAMETERS
6500 GOSUB 14500
6600 REM
6700 REM ESTABLISH NOMINAL RESONANT AND ANTIRESONANT CRYSTAL FREQUENCIES
6800 FS = FIX(1/(2*PI*SQR(L1*C1)))
6900 FA = FIX(1/(2*PI*SQR(L1*C1*CO/(C1+CO))))
7000 PRINT
7100 PRINT "XTAL IS SERIES RESONANT AT ",FS," HZ"
7200 PRINT " PARALLEL RESONANT AT ",FA," HZ"
7300 PRINT
7400 PRINT "SELECT: 1. LIST PARAMETERS"
7500 PRINT " 2. CIRCUIT ANALYSIS"
7600 PRINT " 3. OSCILLATION FREQUENCY"
7700 PRINT " 4. START-UP TIME CONSTANT"
7800 PRINT " 5. STEADY-STATE ANALYSIS"

```

230659-44

```

7900 PRINT
8000 INPUT N
8100 IF N=1 THEN PRINT ELSE 8600
8200 REM
8300 REM ----- LIST PARAMETERS -----
8400 GOSUB 17100
8500 GOTO 8800
8600 IF N=2 THEN PRINT ELSE 9400
8700 REM
8800 REM ----- CIRCUIT ANALYSIS -----
8900 PRINT " FREQUENCY S+JF TYPE (S),(F) "
9000 INPUT SQ,FQ
9100 GOSUB 20200
9200 GOSUB 26600
9300 GOTO 8800
9400 IF N=3 THEN 10300 ELSE 11000
9500 REM
9600 REM ----- OSCILLATION FREQUENCY -----
9700 CL = CX*CY/(CX+CY) + CO
9800 FQ = FIX(1/(2*PI*SQR(L1*CL/(C1+CL))))
9900 SQ = 0
10000 DF = FIX(10*INT(LOG(FA-FS)/LOG(10)-2)+.5)
10100 DS = 0
10200 RETURN
10300 GOSUB 9700
10400 GOSUB 30300
10500 PRINT
10600 PRINT
10700 PRINT "FREQUENCY AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE."
10800 GOSUB 26600
10900 GOTO 8800
11000 IF N=4 THEN PRINT ELSE 12200
11100 REM
11200 REM ----- START-UP TIME CONSTANT -----
11300 PRINT "THIS WILL TAKE SOME TIME"
11400 GOSUB 9700
11500 GOSUB 37700
11600 PRINT
11700 PRINT
11800 PRINT "FREQUENCY AT WHICH LOOP GAIN = 1 AT 0 DEGREES:"
11900 GOSUB 26600
12000 PRINT : PRINT "THIS YIELDS A START-UP TIME CONSTANT OF ";CSNG(1000000/(2*PI*SQ));" MICROSECS"
12100 GOTO 8800
12200 IF N=5 THEN PRINT ELSE 7300
12300 REM
12400 REM ----- STEADY-STATE ANALYSIS -----
12500 PRINT "STEADY-STATE ANALYSIS"
12600 PRINT
12700 PRINT "SELECT: 1. 8031/8051"
12800 PRINT "          2. 8751"
12900 PRINT "          3. 8035/8039/8040/8048/8049"
13000 PRINT "          4. 8748/8749"
13100 INPUT ICX
13200 IF ICX<1 OR ICX>4 THEN 12600
13300 GOSUB 46900
13400 GOTO 7300
13500 REM SUBROUTINE BELOW DEFINES INPUT-OUTPUT CURVE OF OSCILLATOR CKT
13600 IF ICX>2 AND VO=5 AND VI<2 THEN RETURN
13700 VO = -10*VI + 15
13800 IF VO>5 THEN VO = 5
13900 IF VO<2 THEN VO = 2
14000 IF ICX>2 AND VO>2 THEN VO = 5
14100 RETURN
14200 REM
14300 REM *****
14400 REM
14500 REM DEFINE CIRCUIT PARAMETERS
14600 REM
14700 INPUT " R1 (OHMS):",R1
14800 INPUT " L1 (HENRY):",L1
14900 INPUT " C1 (PF):",C1
15000 C1 = X*1E-12
15100 INPUT " CO (PF):",CO
15200 CO = X*1E-12
15300 INPUT " CXTAL1 (PF):",CX
15400 CX = X*1E-12
15500 INPUT " CXTAL2 (PF):",CY
15600 CY = X*1E-12

```

230659-45


```

15700 INPUT " GAIN FACTOR MAGNITUDE";AV#
15800 INPUT " AMP FEEDBACK RESISTANCE (K-OHMS)";X
15900 RX = X*1000#
16000 INPUT " AMP OUTPUT RESISTANCE (K-OHMS)";X
16100 RO = X*1000#
16200 REM
16300 REM
16400 REM      LIST CURRENT PARAMETER VALUES
16500 GOSUB 17100
16600 RETURN
16700 REM
16800 REM
16900 REM *****
17000 REM
17100 REM      LIST CURRENT PARAMETER VALUES
17200 REM
17300 PRINT
17400 PRINT "CURRENT PARAMETER VALUES: 1. R1 = ",R1," OHMS"
17500 PRINT "                                2. L1 = ",CSNG(L1)," HENRY"
17600 PRINT "                                3. C1 = ",CSNG(C1*1E+12)," PF"
17700 PRINT "                                4. CO = ",CSNG(CO*1E+12)," PF"
17800 PRINT "                                5. CTAL1 = ",CSNG(CX*1E+12)," PF"
17900 PRINT "                                6. CTAL2 = ",CSNG(CY*1E+12)," PF"
18000 PRINT "                                7. AMPLIFIER GAIN MAGNITUDE = ",AV#
18100 PRINT "                                8. FEEDBACK RESISTANCE = ",CSNG(RX* .001)," K-OHMS"
18200 PRINT "                                9. OUTPUT RESISTANCE = ",CSNG(RO* .001)," K-OHMS"
18300 PRINT
18400 PRINT "TO CHANGE A PARAMETER VALUE, TYPE (PARAM NO.), (NEW VALUE)."
18500 PRINT "OTHERWISE, TYPE 0,0 "
18600 INPUT NX,X
18700 IF NX=0 THEN RETURN
18800 IF NX=1 THEN R1 = X
18900 IF NX=2 THEN L1 = X
19000 IF NX=3 THEN C1 = X*1E-12
19100 IF NX=4 THEN CO = X*1E-12
19200 IF NX=5 THEN CX = X*1E-12
19300 IF NX=6 THEN CY = X*1E-12
19400 IF NX=7 THEN AV# = X
19500 IF NX=8 THEN RX = X*1000!
19600 IF NX=9 THEN RO = X*1000!
19700 GOTO 17400
19800 REM
19900 REM
20000 REM *****
20100 REM
20200 REM      CIRCUIT ANALYSIS (L+2) YOUNGDAVE
20300 REM
20400 REM      This routine calculates the loop gain at complex frequency SQ+JFQ.
20500 REM
20600 REM      1. Crystal impedance: RE + jXE
20700 REM
20800 X1 = FNXL(FQ,L1) + FNXC(SQ,FQ,C1)
20900 RE = FNRP((R1+FNRL(SQ,L1)+FNRC(SQ,FQ,C1)),X1,FNRC(SQ,FQ,CO),FNXC(SQ,FQ,CO))
21000 XE = FNXP((R1+FNRL(SQ,L1)+FNRC(SQ,FQ,C1)),X1,FNRC(SQ,FQ,CO),FNXC(SQ,FQ,CO))
21100 REM
21200 REM      2. RF + jXF = (RE+jXE):(amplifier feedback resistance)
21300 REM
21400 RF = FNRP(RX,O,RE,XE)
21500 XF = FNXP(RX,O,RE,XE)
21600 REM
21700 REM      3. Input impedance: ZI = RI + jXI = impedance of CTAL1
21800 REM
21900 RI = FNRP(SQ,FQ,CX)
22000 XI = FNXC(SQ,FQ,CX)
22100 REM
22200 REM      4. Load impedance: ZL = (impedance of CTAL2):[(RF+RI)+(XF+XI)]
22300 REM
22400 RL = FNRP((RF+RI),(XF+XI),FNRC(SQ,FQ,CY),FNXC(SQ,FQ,CY))
22500 XL = FNXP((RF+RI),(XF+XI),FNRC(SQ,FQ,CY),FNXC(SQ,FQ,CY))
22600 REM
22700 REM      5. Amplifier gain A = -AV*ZL/(ZL+RO)
22800 REM
22900 REM
23000 AR# = -AV#*FNRP(RL,XL,(RO+RL),XL)
23100 AI# = -AV#*FNXP(RL,XL,(RO+RL),XL)
23200 REM
23300 REM      6. Feedback ratio (beta) = (RI+jXI)/[(RF+RI)+(XF+XI)]
23400 REM

```

71-850053

230659-46

```

23500 REM
23600 BR# = FNRR(RI, XI, (RI+RF), (XI+XF))
23700 BI# = FNXR(RI, XI, (RI+RF), (XI+XF))
23800 REM
23900 REM 7. Amplifier gain in magnitude/phase form: AR+jAI = A at AP degrees
24000 REM
24100 A = FNZM(AR#, AI#)
24200 AP = FNZP(AR#, AI#)
24300 REM
24400 REM 8. (beta) in magnitude/phase form: BR+jBI = B at BP degrees
24500 REM
24600 B = FNZM(BR#, BI#)
24700 BP = FNZP(BR#, BI#)
24800 REM
24900 REM 9. Loop gain G = (BR+jBI)*(AR+jAI)
25000 REM G = G(real) + jG(imaginary)
25100 REM
25200 GR = FNRM(AR#, AI#, BR#, BI#)
25300 GI = FNXM(AR#, AI#, BR#, BI#)
25400 REM
25500 REM 10. Loop gain in magnitude/phase form: GR+jGI = AL at AQ degrees
25600 REM
25700 AL = FNZM(GR, GI)
25800 AQ = FNZP(GR, GI)
25900 RETURN
26000 REM
26100 REM
26200 REM *****
26300 REM
26400 REM PRINT CIRCUIT ANALYSIS RESULTS
26500 REM
26600 PRINT
26700 PRINT " FREQUENCY = ", SQ, " + J", FG, " HZ"
26800 PRINT " XTAL IMPEDANCE = ", FNZM(RE, XE), " OHMS AT ", FNZP(RE, XE), " DEGREES"
26900 PRINT " (RE = ", CSNG(RE), " OHMS)"
27000 PRINT " (XE = ", CSNG(XE), " OHMS)"
27100 PRINT " LOAD IMPEDANCE = ", FNZM(RL, XL), " OHMS AT ", FNZP(RL, XL), " DEGREES"
27200 PRINT " AMPLIFIER GAIN = ", A, " AT ", AP, " DEGREES"
27300 PRINT " FEEDBACK RATIO = ", B, " AT ", BP, " DEGREES"
27400 PRINT " LOOP GAIN = ", AL, " AT ", AQ, " DEGREES"
27500 RETURN
27600 REM
27700 REM
27800 REM *****
27900 REM
28000 REM SEARCH FOR FREQUENCY (S+JF)
28100 REM AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE
28200 REM
28300 REM This routine searches for the frequency at which the imaginary part
28400 REM of the loop gain is zero. The algorithm is as follows:
28500 REM 1. Calculate the sign of the imaginary part of the loop gain (GI).
28600 REM 2. Increment the frequency.
28700 REM 3. Calculate the sign of GI at the incremented frequency.
28800 REM 4. If the sign of GI has not changed, go back to 2.
28900 REM 5. If the sign of GI has changed, and this frequency is within
29000 REM 1Hz of the previous sign-change, exit the routine.
29100 REM 6. Otherwise, divide the frequency increment by -10.
29200 REM 7. Go back to 2.
29300 REM The routine is entered with the starting frequency SQ+jFG and
29400 REM starting increment DS+jDF already defined by the calling program.
29500 REM In actual use either DS or DF is zero, so the routine searches for
29600 REM a GI=0 point by incrementing either SQ or FG while holding the other
29700 REM constant. It returns control to the calling program with the
29800 REM incremented part of the frequency being within 1Hz of the actual
29900 REM GI=0 point.
30000 REM
30100 REM 1. CALCULATE THE SIGN OF THE IMAGINARY PART OF THE LOOP GAIN (GI).
30200 REM
30300 GOSUB 20200
30400 GOSUB 26600
30500 IF GI=0 THEN RETURN
30600 SX# = INT(SGN(GI))
30700 IF SX#=-1 THEN DS = -DS
30800 REM (REVERSAL OF DS FOR GI>0 IS FOR THE POLE-SEARCH ROUTINE.)
30900 REM
31000 REM 2. INCREMENT THE FREQUENCY.
31100 REM
31200 SP = SQ

```

81-88068

230659-47

```

31300 FP = FQ
31400 SQ = SQ + DS
31500 FQ = FQ + DF
31600 REM
31700 REM 3 CALCULATE THE SIGN OF G1 AT THE INCREMENTED FREQUENCY.
31800 REM
31900 GOSUB 20200
32000 GOSUB 26600
32100 IF INT(SGN(G1))=0 THEN RETURN
32200 REM
32300 REM 4. IF THE SIGN OF G1 HAS NOT CHANGED, GO BACK TO 2.
32400 REM
32500 IF SX+INT(SGN(G1))=0 THEN PRINT ELSE 31400
32600 SX = -SX
32700 REM
32800 REM 5 IF THE SIGN OF G1 HAS CHANGED, AND IF THIS FREQUENCY IS WITHIN
32900 REM 1HZ OF THE PREVIOUS SIGN-CHANGE, AND IF G1 IS NEGATIVE, THEN
33000 REM EXIT THE ROUTINE (THE ADDITIONAL REQUIREMENT FOR NEGATIVE G1
33100 REM IS FOR THE POLE-SEARCH ROUTINE.)
33200 REM
33300 IF ABS(SP-SQ)<1 AND ABS(FP-FQ)<1 AND SX=-1 THEN RETURN
33400 REM
33500 REM 6. DIVIDE THE FREQUENCY INCREMENT BY -10
33600 REM
33700 DS = -DS/10#
33800 DF = -DF/10#
33900 REM
34000 REM 7. GO BACK TO 2.
34100 REM
34200 GOTO 31200
34300 REM
34400 REM
34500 REM *****
34600 REM
34700 REM
34800 REM
34900 REM This routine searches for the frequency at which the loop gain = 1
35000 REM at 0 degrees. That frequency is the pole frequency of the closed-
35100 REM loop gain function. The pole frequency is a complex number, SQ+jFQ
35200 REM (Hz). Oscillator start-up ensues if SQ>0. The algorithm is based on
35300 REM the calculated behavior of the phase angle of the loop gain in the
35400 REM region of interest on the complex plane. The locus of points of zero
35500 REM phase angle crosses the j-axis at the oscillation frequency and at
35600 REM some higher frequency. In between these two crossings of the j-axis,
35700 REM the locus lies in Quadrant I of the complex plane, forming an
35800 REM approximate parabola which opens to the left. The basic plan is to
35900 REM follow the locus from where it crosses the j-axis at the oscillation
36000 REM frequency, into Quadrant I, and find the point on that locus where
36100 REM the loop gain has a magnitude of 1. The algorithm is as follows:
36200 REM 1. Find the oscillation frequency, 0+jFQ.
36300 REM 2. At this frequency calculate the sign of (AL-1). (AL = magnitude
36400 REM of loop gain.)
36500 REM 3. Increment FQ.
36600 REM 4. For this value of FQ, find the value of SQ for which the loop
36700 REM gain has zero phase.
36800 REM 5. For this value of SQ+jFQ, calculate the sign of (AL-1).
36900 REM 6. If the sign of (AL-1) has not changed, go back to 3.
37000 REM 7. If the sign of (AL-1) has changed, and this value of FQ is
37100 REM within 1Hz of the previous sign-change, exit the routine.
37200 REM 8. Otherwise, divide the FQ-increment by -10.
37300 REM 9. Go back to 3.
37400 REM
37500 REM 1. FIND THE OSCILLATION FREQUENCY, 0+jFQ.
37600 REM
37700 GOSUB 9700
37800 GOSUB 30300
37900 REM
38000 REM 2. AT THIS FREQUENCY, CALCULATE THE SIGN OF (AL-1).
38100 REM
38200 SYX = INT(SGN(AL-1))
38300 IF SYX=-1 THEN STOP
38400 REM ESTABLISH INITIAL INCREMENTATION VALUE FOR FQ
38500 F1 = FQ
38600 DF = (FA-F1)/10#
38700 GOSUB 30300
38800 DE = (FG-F1)/10#
38900 DF = 0
39000 FG = F1

```

230659-48

```

39100 REM
39200 REM 3. INCREMENT FQ.
39300 REM
39400 FQ = FQ + DE
39500 REM
39600 REM 4. FOR THIS VALUE OF FQ, FIND THE VALUE OF SQ FOR WHICH THE LOOP
39700 REM GAIN HAS ZERO PHASE. (THE ROUTINE WHICH DOES THAT NEEDS DF = 0.
39800 REM SO THAT IT CAN HOLD FQ CONSTANT, AND NEEDS AN INITIAL VALUE FOR
39900 REM DS, WHICH IS ARBITRARILY SET TO DS = 1000.)
40000 REM
40100 DS = 1000#
40200 SQ = 0
40300 GOSUB 30300
40400 IF AL=1! THEN RETURN
40500 REM
40600 REM 5. FOR THIS VALUE OF SQ+JFQ, CALCULATE THE SIGN OF (AL-1).
40700 REM 6. IF THE SIGN OF (AL-1) HAS NOT CHANGED, GO BACK TO 3.
40800 REM
40900 IF SYX+INT(SQ*(AL-1))=0 THEN PRINT ELSE 39400
41000 REM
41100 REM 7. IF THE SIGN OF (AL-1) HAS CHANGED, AND THIS VALUE OF FQ IS WITHIN
41200 REM 1HZ OF THE PREVIOUS SIGN-CHANGE, EXIT THE ROUTINE.
41300 REM
41400 IF ABS(F1-FQ)<1 THEN RETURN
41500 REM
41600 REM 8. DIVIDE THE FQ-INCREMENT BY -10.
41700 REM
41800 DE = -DE/10#
41900 F1 = FQ
42000 SYX = -SYX
42100 REM
42200 REM 9. GO BACK TO 3.
42300 REM
42400 GOTO 39400
42500 REM
42600 REM
42700 REM *****
42800 REM
42900 REM STEADY-STATE ANALYSIS
43000 REM
43100 REM The circuit model used in this analysis is similar to the one used
43200 REM in the small-signal analysis, but differs from it in two respects.
43300 REM First, it includes clamping and clipping effects described in the
43400 REM text. Second, the voltage source in the Thevenin equivalent of the
43500 REM amplifier is controlled by the input voltage in accordance with an
43600 REM input-output curve defined elsewhere in the program.
43700 REM The analysis applies a sinusoidal input signal of arbitrary
43800 REM amplitude, at the oscillation frequency, to the XTAL1 pin, then
43900 REM calculates the resulting waveform from the voltage source. Using
44000 REM standard Fourier techniques, the fundamental frequency component of
44100 REM this waveform is extracted. This frequency component is then
44200 REM multiplied by the factor :ZL/(ZL+RO):, and the result is taken to be
44300 REM the signal appearing at the XTAL2 pin. This signal is then
44400 REM multiplied by the feedback ratio (beta), and the result is taken to
44500 REM be the signal appearing at the XTAL1 pin. The algorithm is now
44600 REM repeated using this computed XTAL1 signal as the assumed input
44700 REM sinusoid. Every time the algorithm is repeated, new values appear at
44800 REM XTAL1 and XTAL2, but the values change less and less with each
44900 REM repetition. Eventually they stop changing. This is the steady-state.
45000 REM The algorithm is as follows:
45100 REM 1. Compute approximate oscillation frequency.
45200 REM 2. Call a circuit analysis at this frequency.
45300 REM 3. Find the quiescent levels at XTAL1 and XTAL2 (to establish the
45400 REM beginning DC level at XTAL1).
45500 REM 4. Assume an initial amplitude for the XTAL1 signal.
45600 REM 5. Correct the DC level at XTAL1 for clamping effects, if necessary.
45700 REM 6. Using the appropriate input-output curve, extract a DC level and
45800 REM the fundamental frequency component (multiplying the latter by
45900 REM :ZL/(ZL+RO):).
46000 REM 7. Clip off the negative portion of this output signal, if the
46100 REM negative peak falls below zero.
46200 REM 8. If this signal, multiplied by (beta), differs from the input
46300 REM amplitude by less than 1mV, or if the algorithm has been repeated
46400 REM 10 times, exit the routine
46500 REM 9. Otherwise, multiply the XTAL2 amplitude by (beta) and feed it
46600 REM back to XTAL1, and go back to 5
46700 REM
46800 REM 1. COMPUTE APPROXIMATE OSCILLATION FREQUENCY.

```



```

46900 GOSUB 9700
47000 REM
47100 REM      2. CALL A CIRCUIT ANALYSIS AT THIS FREQUENCY.
47200 GOSUB 20800
47300 PRINT : PRINT      PRINT "ASSUMED OSCILLATION FREQUENCY:"
47400 GOSUB 26600
47500 PRINT : PRINT
47600 REM
47700 REM      3. FIND QUIESCENT POINT
47800 REM (At quiescence the voltages at XTAL1 and XTAL2 are equal. This
47900 REM voltage level is found by trial-and-error, based on the input-
48000 REM output curve, so that a person can change the input-output curve
48100 REM as desired without having to re-calculate the quiescent point.)
48200 VI = 0
48300 VB = 1
48400 K1 = 1
48500 VI = VI + VB
48600 GOSUB 13600
48700 IF ABS(V0-VI)<.001 THEN 49200
48800 IF K1+SGN(V0-VI)=0 THEN 48900 ELSE 48500
48900 K1 = SGN(V0-VI)
49000 VB = -VB/10
49100 GOTO 48500
49200 VB = VI
49300 PRINT "QUIESCENT POINT = ";VB
49400 REM
49500 REM      4. ASSUME AN INITIAL AMPLITUDE FOR THE XTAL1 SIGNAL.
49600 EI = .01
49700 NRX = 0
49800 REM
49900 REM      5. CORRECT FOR CLAMPING EFFECTS, IF NECESSARY.
50000 REM (K1 and K2 are curve-fitting parameters for the ROM parts.)
50100 K1 = (2.5-VB)/(3-VB)
50200 K2 = (VB-1.25)/(3-VB)
50300 IF ICX=2 OR ICX=4 THEN IF EI<(VB+.5) THEN EO = VB ELSE EO = EI - .5
50400 IF ICX=1 OR ICX=3 THEN IF EI<(VB+.5) THEN EO = VB ELSE EO = K1*EI+K2
50500 NRX = NRX + 1
50600 REM
50700 REM      6. DERIVE XTAL2 AMPLITUDE
50800 V0 = 0
50900 VC = 0
51000 VS = 0
51100 FOR NX = -25 TO +24
51200 VI = EO - EI*COS(PI*NX/25)
51300 GOSUB 13600
51400 V0 = V0 + VI
51500 VC = VC + V0*COS(PI*NX/25)
51600 VS = VS + V0*SIN(PI*NX/25)
51700 NEXT NX
51800 V0 = V0/50
51900 V1 = SQR(VC^2+VS^2)/25*FNZM(RL,XL)/FNZM((RL+RD),XL)
52000 REM
52100 REM      7. CLIP XTAL2 SIGNAL
52200 IF V0-V1<0 THEN VL = 0 ELSE VL = V0-V1
52300 PRINT : PRINT "XTAL1 SWING = ";EO-EI;" TO ";EO+EI
52400 PRINT "XTAL2 SWING = ";VL;" TO ";V0+V1
52500 REM
52600 REM      8. TEST FOR TERMINATION
52700 IF ABS(EI-V1*B)<.001 OR NRX=10 THEN RETURN
52800 REM
52900 REM      9. FEED BACK TO XTAL1 AND REPEAT
53000 EI = V1*B
53100 GOTO 50300

```

230659-50

